Access Control to Information in Pervasive Computing Environments

Urs Hengartner

CMU-CS-05-160

August 2005

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Peter Steenkiste, Chair
Adrian Perrig
Michael K. Reiter
Edward W. Felten, Princeton

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the Army Research Office under contract no. DAAD19-02-1-0389, the US Navy under grant no. N66001992891801, and the US Air Force Research Laboratory under grant no. F306029910518. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
AUG 2005		**00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Access Control to Information in Pervasive Computing Environments	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**217**	

Keywords: Information security, privacy, semantics of information, derivation of information, context awareness, applied cryptography, identity-based encryption.

Abstract

Pervasive computing envisions a world in which we are surrounded by embedded, networked devices, which gather and share information about people, such as their location, activity, or even their feelings. Some of this information is confidential and should be released only to authorized entities. In this thesis, I show how existing solutions for controlling access to information are not sufficient for pervasive computing. In particular, there are four challenges: First, there will be many information services, run by different organizations, even in a single social environment, which makes centralized access control infeasible and authorization management difficult. Second, there will be complex types of information, such as a person's calendar entry, which could leak other kinds of information, such as the person's current location. Third, there will be services that derive specific information, such as a person's activity, from raw information, such as a videostream, and that become attractive targets for intruders. Fourth, access decisions could be constrained based on confidential information about an individual's context and could leak this confidential information.

This thesis presents a distributed access-control architecture for pervasive computing that supports complex and derived information and confidential context-sensitive constraints. Namely, the thesis makes the following contributions: First, I introduce a distributed access-control architecture, in which a client proves to a service that the client is authorized to access the requested information. Second, I incorporate the semantics of complex information as a first-class citizen into this architecture, based on *information relationships*. Third, I propose *derivation-constrained access control*, which reduces the influence of intruders into a service by making the service prove that it is accessing information on behalf of an authorized client. Fourth, I study the kinds of information leaks that confidential context-sensitive constraints can cause, and I introduce *access-rights graphs* and *hidden constraints* to address these leaks. Fifth, I present *obscured proof-of-access descriptions*, which allow a service to inform a client of the required proof of access without leaking confidential information being part of this description. Sixth, as an alternative approach, I introduce an encryption-based access-control architecture for pervasive computing, in which a service gives information to any client, but only in an encrypted form.

Acknowledgments

I would like to thank

my advisor, Peter Steenkiste, for his guidance and support,

my other committee members, Ed Felten, Adrian Perrig, and Mike Reiter, for their feedback,

Lujo Bauer, Nick Hopper, and Dawn Song for their input,

Thomas Gross for inviting me to Carnegie Mellon,

Mahim, Mukesh, and Rajesh for their friendship, and

Kate for her love and support.

Contents

1 Introduction **1**

1.1 Privacy in Pervasive Computing 1

1.1.1 Diversity in Service Administration 4

1.1.2 Complex Information 5

1.1.3 Derivation of Information 6

1.1.4 Confidential Context-Sensitive Constraints 7

1.2 Related Work 8

1.2.1 Diversity in Service Administration 8

1.2.2 Complex Information 9

1.2.3 Derivation of Information 9

1.2.4 Confidential Context-Sensitive Constraints 9

1.3 Thesis Goal 10

1.4 Contributions 11

1.5 Guide To The Thesis 12

2 Design of Access-Control Architecture **15**

2.1 Overview 15

2.2 Access Policies 16

2.2.1 Properties of Access Policies 17

2.2.2 Access Rights 19

2.3 Access-Control Architectures 21

2.3.1 Centralized Access Control 21
2.3.2 Service-based Access Control 22
2.3.3 Client-Based Access Control 22
2.3.4 End-to-End Access Control 23
2.4 People-Location System 25
2.4.1 Overview 25
2.4.2 Access Rights 25
2.4.3 Distributed Access Control 28
2.4.4 Lessons 30
2.5 Access Rights 32
2.5.1 Information Representation 32
2.5.2 Formal Model 35
2.5.3 Implementation 37
2.6 Distributed Access Control 38
2.6.1 Access-Rights Storage 38
2.6.2 Proof Verification 39
2.6.3 Implementation 39
2.6.4 Deployment 40
2.7 Security Model 41
2.8 Related Work 42
2.8.1 People-Location Systems 42
2.8.2 Information-Representation Schemes 43
2.8.3 Access-Control Logics 44
2.8.4 Trust-Management Systems 45
2.9 Summary 45

3 Information Relationships **47**
3.1 Overview 47
3.2 Information Relationships 49

3.3 Formal Model 51
3.3.1 Bundling-Based Relationships 52
3.3.2 Combination-Based Relationships 53
3.3.3 Granularity-Based Relationships 54
3.3.4 Example 55
3.4 Discussion 56
3.5 Global Information Relationships 58
3.5.1 Delegation-Based Approach 58
3.5.2 Naming-Based Approach 60
3.6 Proof Building 61
3.6.1 Bundling-Based Relationships 61
3.6.2 Combination-Based Relationships 63
3.6.3 Granularity-Based Relationships 64
3.7 Implementation 64
3.8 Performance Analysis 65
3.8.1 Number of Access Rights 65
3.8.2 Proof-Building Time 67
3.8.3 Client-Response Time 69
3.9 Proof-Carrying Authorization 70
3.9.1 Overview 71
3.9.2 Bundling-Based Relationships 71
3.9.3 Combination-Based Relationships 73
3.9.4 Granularity-Based Relationships 74
3.9.5 Discussion 74
3.10 Related Work 77
3.11 Summary 78

4 Derivation-Constrained Access Control **81**
4.1 Overview 82

4.2 Background . . . 83
4.3 Derivation-Constrained Access Control . . . 85
4.3.1 Derivation-Constrained Access Rights . . . 85
4.3.2 Derivation Properties . . . 85
4.3.3 Access Control . . . 86
4.4 Formal Model . . . 87
4.5 Examples . . . 89
4.5.1 People Location . . . 89
4.5.2 Health Information . . . 91
4.6 Security Analysis . . . 92
4.7 Implementation . . . 94
4.8 Performance Analysis . . . 95
4.8.1 Measurements . . . 96
4.8.2 Discussion . . . 98
4.9 Related Work . . . 98
4.10 Summary . . . 100

5 Confidential Context-Sensitive Constraints 101
5.1 Overview . . . 101
5.2 System Model . . . 103
5.2.1 Nature of Constraints . . . 103
5.2.2 Scenario . . . 103
5.2.3 Enforceability . . . 104
5.2.4 Staleness of Constraint Information . . . 105
5.3 Formal Model . . . 106
5.4 Constraints and Information Leaks . . . 109
5.4.1 Centralized Access Control . . . 110
5.4.2 Service-Based Access Control . . . 113
5.4.3 Client-Based Access Control . . . 115

5.4.4 Third Entity 116
5.5 Access-Rights Graphs 117
5.5.1 Design 117
5.5.2 Centralized Access Control 120
5.5.3 Service-Based Access Control 121
5.5.4 Client-Based Access Control 123
5.6 Hidden Constraints 124
5.6.1 Design 124
5.6.2 Formal Model 126
5.6.3 Discussion 126
5.7 Constraints and Information Relationships 127
5.8 Architecture 127
5.8.1 Overview 127
5.8.2 Hidden Constraints 128
5.8.3 Digital Certificates 132
5.9 Performance Analysis 134
5.10 Related Work 137
5.11 Summary 139

6 Obscured Proof-of-Access Descriptions 141
6.1 Overview 141
6.2 Requirements 142
6.3 Obscured Proof-of-Access Descriptions 144
6.3.1 Hierarchical Identity-Based Encryption 144
6.3.2 Basic Operations 145
6.3.3 Architecture 146
6.3.4 Limiting the Search Space 151
6.4 Discussion 152
6.4.1 Identity-Based Encryption 152

6.4.2 Forwarded Access Rights 154
6.4.3 Information Relationships 155
6.4.4 Confidential Constraints 156
6.5 Prototype Implementation 158
6.6 Related Work 159
6.7 Summary 162

7 Encryption-Based Access Control 163
7.1 Overview 163
7.2 Requirements 164
7.3 Encryption-Based Access Control 165
7.4 Complex Information 166
7.5 Prototype Implementation 167
7.6 Evaluation 168
7.7 Related Work 170
7.8 Summary 172

8 Conclusions and Future Work 173
8.1 Contributions 173
8.2 Reflections 175
8.2.1 Office Scenario 175
8.2.2 Home Scenario 177
8.3 Directions for Future Work 179
8.3.1 Remote Discovery of Certificates 179
8.3.2 Semantics for Access-Control Logic 179
8.3.3 Semantic-Web Concepts 180
8.3.4 Uncertainty in Access Control 180
8.4 Summary 180

A Hierarchical Identity-Based Encryption 183

A.1 Operations 183
A.2 Optimizations 185
A.3 Evaluation 185

Bibliography **187**

List of Figures

1.1 Multitude of information services. 2

2.1 Information gateways. 23
2.2 People-location system. 26
2.3 SPKI/SDSI certificate. 27
2.4 Extended SPKI/SDSI certificate. 38

3.1 Structured proof. 62
3.2 Proof building. 63
3.3 Bundling-based information relationship. 65
3.4 Number of issued statements. 67
3.5 Proof-building time. 68
3.6 Client-response time. 70

4.1 Gateway. 82
4.2 Derivation property and derivation-constrained access right. 94

5.1 Information leak to issuer of access right. 112
5.2 Access-rights graph. 118
5.3 Building of access-rights graphs. 119
5.4 Access-rights graphs with conflict. 120
5.5 Access-control algorithm. 122
5.6 Proof-building architecture. 128

5.7 One-way chain. 130
5.8 Extended SPKI/SDSI certificates. 133
5.9 Hidden, one-way chain-based constraints. 136

6.1 Architecture for client-based access control. 146
6.2 Hierarchies. 147
6.3 Bundling-based information hierarchy. 155
6.4 Location hierarchy. 156

7.1 Architecture for encryption-based access control. 165
7.2 Performance of encryption/decryption. 169

List of Tables

3.1 Example statements. 55
3.2 Client-response time. 69
3.3 Proof of Axiom 3.11. 73
3.4 Proof of Axiom 3.12. 75

4.1 Statements in the people-location scenario. 90
4.2 Client-response time. 96

5.1 Information leaks. 127
5.2 Client-response time. 136

6.1 Key-management demand. 152

A.1 Processing times. 186

Chapter 1

Introduction

Pervasive computing envisions a world in which hundreds of wireless computing devices are available to a person. These devices will be embedded in the environment so that people will use them without thinking about them [127]. This infrastructure enables the continuous gathering and sharing of information about people. For example, there will be sensors for monitoring people's location, activity, health status, or even feelings.

Some of these sensors have already started showing up in our daily lives. For instance, a lot of places are monitored by webcams connected to the Internet or by surveillance cameras. Teen Arrive Alive [6] allows parents to track their children, based on GPS-equipped cellphones from Nextel [99]. BodyMedia [23] offers an armband that monitors medical information, such as skin temperature and movement, and that communicates this information to a base station using wireless technology.

The enormous amounts of personal information gathered in pervasive computing make privacy a major concern [80, 82]. Most personal information is confidential, that is, not just anyone should be able to access it. Therefore, a pervasive computing environment must control access to confidential personal information.

1.1 Privacy in Pervasive Computing

To get a better understanding of the challenges for access control to confidential personal information in pervasive computing, let us look at an example scenario. The scenario illustrates how personal information is gathered, provided, and used in pervasive computing. Figure 1.1 presents the providers of information that participate in the scenario. We

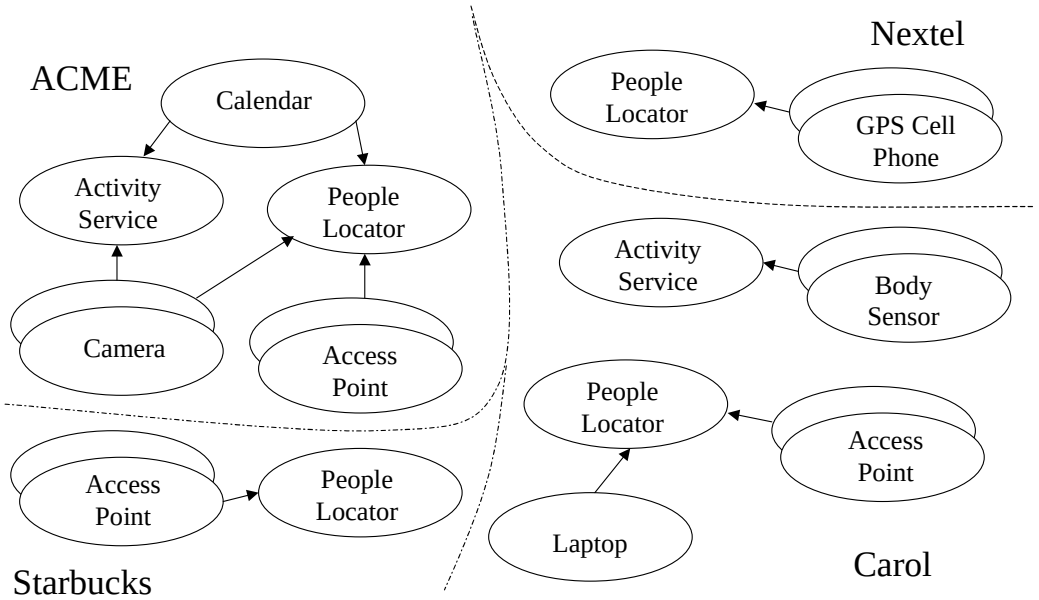


Figure 1.1: Multitude of information services. The ovals represent providers of information ("services"). The arrows indicate information flows. The services are maintained by different organizations.

use the term *service* for such a provider. In the scenario, Carol, who works for company ACME, uses a centralized calendar application, maintained by her employer. For example, she schedules a meeting with Bob that will take place in her office. Carol lets her executive assistant, Alice, access her current calendar entry. However, Carol limits this access such that Alice will be granted access only if Carol is in Pittsburgh and if Carol is not vacationing. Therefore, whenever Alice wants to access Carol's current calendar entry, the mechanism controlling access to this information needs to retrieve Carol's current location and activity information from services offering this information and validate the restrictions imposed by Carol.

Multiple services, exploiting different technologies, provide Carol's location information. For example, ACME deploys cameras in the company's meeting rooms and provides a wireless network. Therefore, ACME can locate Carol by detecting her face in a camera's videostream or by finding the access point to which Carol's Wi-Fi laptop is connecting. Furthermore, Carol's cellphone company, Nextel, makes her location information available. Finally, while Alice is at home or at Starbucks, her and Starbucks' wireless network, respectively, also offer her location information. Carol's activity information is also provided by multiple services. For example, one of the services derives this information from information gathered by body sensors worn by Carol.

Our scenario reveals four interesting properties of information in pervasive computing:

- Information will be offered by multiple services, potentially run by different organizations, even in a single social environment. For example, while Carol is at work, both Carol's cellphone company and her employer provide her location information. Also, different social environments might exploit different services. While Carol is at work, her employer's location service will provide her location information. While she is at home, her own location service will offer this information. In both environments, the location service of her cellphone company could provide additional location information.

- Certain types of information will be *complex*. By complex information, we mean information that reveals other types of information. For example, Carol's current calendar entry about her meeting with Bob in her office reveals information about Carol's current location and activity. In addition, the entry also gives away Bob's current location and activity.

- Information will be processed and will change its nature while flowing from its source through multiple services to a client asking for information. For example, one of the location services retrieves a videostream from a camera, which is also a

service, detects Carol's face in the stream, and determines Carol's current location. Another location service derives Carol's location from her calendar entry, provided by the calendar application.

- Information will be retrieved from a service not only for handing it over to a client asking for this information (or for information derived from this information), but also for constraining access to information that the client asked for. For example, Carol's location information is used for constraining Alice's access to Carol's calendar information.

Access control to information has been well investigated in the context of information stored in distributed and remote filesystems (such as AFS [67] with Kerberos [118], SFS [90], or Microsoft Windows 2000 Server [92]) or in relational databases (such as System R [55], Oracle Database [101], or databases exploiting role-based access control [100]). However, the properties of information that we just discussed make access control in pervasive computing challenging, and applying existing access-control techniques could easily result in information leaks [60]. Let us take a closer look at the challenges and how they have been traditionally addressed for filesystems or databases. We will summarize each challenge in a lesson for access control to information in pervasive computing.

1.1.1 Diversity in Service Administration

Services that provide information within a social environment can be administrated by different organizations. For example, while Carol is at work, her cellphone company offers her location information, and the calendar application run by her employer provides her calendar information. An organization might not be authorized to access information provided by another organization. Therefore, access control is required when services exchange information with each other. For example, assume that the calendar application contacts the location service run by Carol's cellphone company in order to ensure that Carol is currently in Pittsburgh, as required by Carol when she granted Alice access to her current calendar entry. Here, the cellphone company must validate that the calendar application (i.e., the organization running this service) has access to Carol's location information. In contrast, within a specific social environment, a distributed filesystem or a database is typically deployed by a single organization on a set of machines that are trusted by the organization and that communicate over secure channels. In this setting, access control is required only at the interface between the trusted machines and entities accessing information provided by the filesystem or database. (Such an entity is typically

a machine or an application that accesses information on a user's behalf.) There is no need to run access control (other than authentication) when file servers exchange information or while information is processed in a distributed database.

Not only can there be services run by different organizations in a single social environment, there can also be multiple social environments. If people transitioned between multiple environments, this transitioning should be seamless. For instance, it should not be necessary for an individual to decide whom to grant access to her location information for each environment. Instead, the person should be able to make a single decision that applies to any environment in which she participates. In the case of filesystems or databases, these decisions are typically coupled to the organization that provides information in an environment, and sharing the same decision across environments is not possible. For example, in a remote filesystem, such as AFS [67], an individual needs to make separate decisions for each set of directories or files bound to a particular organization that is part of the filesystem. An individual can have a home directory both at Princeton and at Carnegie Mellon, but she cannot make a single decision that applies to her private files or directories both at Princeton and at Carnegie Mellon.

The lesson from this challenge is that, in pervasive computing, access control needs to be able to deal with services run by different entities, while making it easy for individuals to manage access to their personal information provided by these services.

1.1.2 Complex Information

Complex information enables people learning this information to derive multiple types of information about one or multiple individuals. For example, a calendar entry can reveal location and activity information. Similarly, a videostream delivered by a camera can reveal the location and activity of the people shown in the videostream. If a client that accesses this complex information is not authorized to access the derivable information, there will be an information leak and a person's privacy will be invaded.

Such information leaks are also possible in filesystems or databases. Avoiding these leaks in a filesystem is hard since (current) filesystems are not aware of the contents of a file; access control takes only the name of a file into account. A coarse-grained way for an individual to protect a file that contains complex information (e.g., a schedule) is to keep the file private and to not put anyone else on the access control list ("ACL") of the file. Obviously, this solution does not work for pervasive computing, where information needs to be accessible in order to serve people. Alternatively, the individual could carefully establish the ACL such that a client is put on the ACL only if the client has access to all

the information revealed by the complex information in the file. However, this process is tedious and error prone and could lead to consistency problems. For example, consistency problems arise if the revocation of a client's access to information revealed by the complex information stored in the file does not result in the removal of the client from the ACL of the file.

Similarly, a database can provide complex information in the rows of a table (or of a view derived from tables). Database access control is based on the name of a table (or view) and can be row based [101].[1] To prevent a row from leaking information, the entity that grants access to the row must be aware of the clients that have access to all the information stored in the row and grant only these clients access to the row. However, this approach suffers from the same problems as the ones mentioned in the filesystem scenario.

The lesson from this challenge is that, in pervasive computing, access control itself needs to be aware of the contents of complex information and treat this content as a first-class citizen when making an access decision.

1.1.3 Derivation of Information

In pervasive computing, there will be services that derive information. In particular, these services will take *input information* and derive *output information*. Input information can be raw, such as a videostream, audiostream, or a person's heart rate, and potentially reveal information in addition to the output information derived by a service. For example, while a service can derive location information from a videostream, it might also be possible to derive activity or health information from the stream. If a service that derives information has access to raw information, it will become an attractive target for attackers. Namely, an intruder into the service will be able to access the raw information.

In a filesystem, there is no processing of information. In a database, information can be processed (e.g., a view can contain processed information). However, as mentioned in Section 1.1.1, the information gathering and processing typically take place within the trusted computing base established by the organization responsible for the database. As a result, an intruder into the trusted computing base will automatically have access to input information. In pervasive computing, services do not necessarily trust each other and are not part of a single trusted computing base, which gives us the opportunity to implement more sophisticated access control and to limit intruders' access to input information.

[1]We ignore column-based access control, where the values in some columns are masked, since this selective disclosure of complex information can lead to information leaks based on statistical inference, see Section 3.4.

The lesson from this challenge is that, in pervasive computing, a service's access to input information should be limited such that the service can still perform its derivation functionality, but without giving an intruder into the service complete access to the input information.

1.1.4 Confidential Context-Sensitive Constraints

The availability of an individual's context-sensitive information (such as her current location or the current time) makes it possible to exploit this information for context-sensitive access control. For example, in our example scenario, Carol grants access to Alice only if Carol is in Pittsburgh and if Carol is not vacationing. However, exploiting confidential context-sensitive information in a constraint can result in information leaks. In particular, by observing the fate of a request for information that is accessible only under a set of context-sensitive constraints, an observer that knows about the nature of these constraints will learn whether they are satisfied. If any of these constraints involved confidential information to which the observer does not have access, there could be an information leak. For example, if Alice is granted access to Carol's calendar entry, she will learn that Carol is in Pittsburgh, regardless whether Alice is authorized to access Carol's location information.

Current filesystems or databases are limited in the types of definable constraints or do not exploit confidential context-sensitive constraints. For example, filesystems or databases typically limit constraints to membership in a group or role, respectively. For some databases [101], it is possible to constrain access based on context-sensitive information, such as the current time. However, this information is not confidential and knowing it does not result in an information leak. While it is possible to incorporate more flexible context-sensitive constraints into database access control, this option has received limited interest because of the missing availability of context-sensitive information (such as location information). However, pervasive computing is going to make such information much more widely available.

The lesson from this challenge is that, in pervasive computing, access control needs to support context-sensitive constraints, but without leaking the potentially confidential information listed in such a constraint.

1.2 Related Work

In this thesis, we present an access-control architecture that solves the challenges discussed in the previous section. Before we outline our approach, let us review the solutions chosen by other research projects in pervasive computing. We will re-visit each of the challenges, and will see that none of them has been addressed sufficiently in previous work.

1.2.1 Diversity in Service Administration

Many pervasive computing projects [5, 38, 42, 96, 132, 121, 75] rely on the existence of a centralized entity (e.g., "inference engine" [5], "specialized server entity"[38], "security agent" [75], or "middleware service"[96]) that knows all the services in an environment and that hides the fact that multiple services can provide the same type of information from individuals. The centralized entity is defined as being trusted by all the services to run access control on their behalf and as being able to retrieve information from any service.

The advantage of this approach is that it prevents individuals from having to be aware of each service. In particular, individuals do not have to decide for each service whom to grant access to the information provided by the service.

However, this approach has several drawbacks. The existing projects typically assume that all services within an environment are run by the organization associated with the environment. However, this assumption does not necessarily hold in practice. For example, in our example scenario, Carol might want to use the location service provided by her cellphone company regardless of the environment that she is participating in. In addition, many research projects have been deployed only in the context of a single environment; it is not clear how they would be deployed in multiple environments. Furthermore, while the approach prevents an individual from making service-dependent decisions about whom to grant access, the individual still has to decide for each type of information whom to grant access. This approach will not scale if there are lots of different types of information.

There are a few projects that are not centralized [93, 66, 51]. However, these projects assume that an individual controls all the services that provide information about her [51, 66], which is not the case in practice, or they do not address the scalability problem [93], where multiple services offer the same information.

1.2.2 Complex Information

Many existing pervasive computing projects initially neglected access control to information and focused on "Smart Rooms" [75] and on controlling access to physical resources available in such a room, such as a projector or a printer [5, 42, 75]. However, the upcoming of the Semantic Web [17] has also triggered interest in access control to information. The goal of the Semantic Web is to develop languages that are adequate for representing and reasoning about the semantics of information on the Web. Several projects [38, 51] have exploited techniques from the Semantic Web for representing and reasoning about individuals' privacy preferences and about resources in a pervasive computing environment. However, while it has been recognized that complex information can cause information leaks [38], the problem has not been addressed in a deployed architecture.

1.2.3 Derivation of Information

As discussed in Section 1.2.1, many existing pervasive computing projects rely on a centralized approach for running access control. Therefore, in terms of dealing with derived information, they suffer from the same problem as traditional solutions for access control in a filesystem or database, as explained in Section 1.1.3. In particular, the information gathering and derivation take place within a trusted computing base, and an intruder automatically gets access to the input information. Projects that are not centralized and that address information derivation [93] do not suffer from this drawback. However, it is still possible for an intruder into a service to submit authorized queries for input information that the service can access to other services.

1.2.4 Confidential Context-Sensitive Constraints

In existing pervasive computing projects [5, 38, 51, 93], it is typically possible to constrain an individual's access to a resource based on the individual's context, such as her location or the current time. However, most projects ignore information leaks caused by such context-sensitive constraints. While there has been some research in this area [93], the existing research considers only a small subset of the information leaks that constraints could cause in pervasive computing. We will explore this limitation in more detail in Section 5.10.

1.3 Thesis Goal

The challenges and the related work discussed in Sections 1.1 and 1.2, respectively, raise the following central question:

> *Is it possible to control access to confidential information in pervasive computing environments, where*
>
> - *services providing this information are maintained by different administrative entities,*
> - *the offered information can be of a complex nature and reveal other types of information, and*
> - *access decisions can be constrained based on confidential context-sensitive information about an individual,*
>
> *without*
>
> - *requiring individuals to manage their personal information in a service or environment-specific way,*
> - *relying on a trusted centralized authority,*
> - *leaking confidential information revealed by complex information, and*
> - *leaking confidential information used in context-sensitive constraints?*

In this thesis, we answer this question in an affirmative way. We build on the following key techniques:

Client-based access control. Since services in pervasive computing environments will be administrated by different organizations, we cannot expect these services to trust a single centralized entity to run access control on their behalf. Instead, we adopt a distributed approach, where services themselves control access to their provided information, assisted by clients. In particular, a client needs to prove to a service that the client is authorized to access the requested information, based on the possession of one or multiple digital certificates. Such *client-based* access-control architectures are not a new concept. However, existing implementations [16, 69] of the concept are not targeted at pervasive computing and do not deal with the challenges mentioned in Section 1.1. In addition, for complex information, they can leak information when a service informs a client of the required proof of access. Our architecture is based on a formal model for access control in a distributed system.

Flexible information representation. Since there will be many services offering information and many different types of information, we need to minimize the number of decisions that an individual needs to make when deciding about whom to grant access to her personal information. We introduce a representation scheme for information that allows an individual to grant service-independent and environment-independent access to her personal information and to re-use the same decision for multiple types of information.

Semantics of information. Since complex information can reveal multiple types of information, access control must be aware of the semantics of information. We formalize the semantics of information and incorporate these semantics as a first-class citizen into a formal model of access control. In particular, our formal model supports access control to complex information and to derived information. Based on this model, we make a client-based access-control architecture for pervasive computing aware of the semantics of the information to which access is controlled.

1.4 Contributions

All the contributions of this thesis have been implemented in an actual pervasive computing environment and evaluated with measurements or analytically. In particular, this thesis makes the following key contributions:

- We present a client-based access-control architecture for pervasive computing that supports complex and derived information and context-sensitive access decisions. Our distributed approach simplifies dealing with services run by different organizations.

- We introduce the concept of *information relationships*, which capture the semantics of complex information and allow us to incorporate the semantics as a first-class citizen into our client-based access-control architecture. Information relationships can prevent information leaks caused by complex information revealing other types of information. Furthermore, information relationships can make access management easier for individuals since they allow an individual to apply a single access-management statement to multiple types of information.

- We present the concept of *derivation-constrained access control*, which enables our client-based access-control architecture to take derivation properties of information into account for access control. Namely, the concept requires a service that derives

information to prove that the service asks for input information on behalf of a client authorized to access the derived information. In this way, we can limit the influence of intruders into a service that derives information.

- We present the concept of *access-rights graphs*, which supports the identification of potential information leaks in context-sensitive access control and simplifies resolution of constraints. Furthermore, we introduce the concept of *hidden constraints*, which can prevent context-sensitive constraints from leaking confidential information. Finally, we show how to incorporate the concepts into our client-based access-control architecture.

- We present the concept of *obscured proof-of-access descriptions*. These descriptions address information leaks that are possible when a service that provides complex information informs a client of the required proof of access. We also show how to incorporate the concept into our client-based access-control architecture.

- As an alternative option to distributed access control based on proofs of access, we present an encryption-based access-control architecture for pervasive computing. In this architecture, a service grants any client access to its information, but the service encrypts the information before giving the information to a client.

Our access-control architecture does not address the following challenges:

- Our architecture controls read access to information. It does not control write access to information. Similarly, it does not provide access control to resources other than information (e.g., a projector or a printer).

- Our architectures does not provide service localization, that is, locating the service that provides information about an individual in a set of services. We do address a subset of this problem in the context of context-sensitive constraints.

- Proofs of access in client-based access control contain digital certificates showing that an individual grants another individual access to information. Our architecture does not address remote discovery of certificates. We assume that a client that needs to assemble a proof of access has the necessary certificates locally available.

1.5 Guide To The Thesis

Chapter 2, apart from Section 2.4, is fundamental to the rest of the thesis and should be read first. The reader does not have to follow the other chapters in order, they are mostly

independent of each other. The only exception is Chapter 7, which depends on Chapter 6. In more detail, the rest of this thesis is organized as follows:

In Chapter 2, we introduce the concepts of access policies and access rights and discuss the application of these concepts in the context of a people-location system and of a more general, client-based access-control architecture for pervasive computing. As a part of the latter architecture, we present an information-representation scheme and a formal model for access control. We will use the access-control architecture as a base for incorporating the contributions made in the other chapters.

In Chapter 3, we introduce information relationships and add support for them to the formal model and to the client-based access-control architecture. We also provide an analytical and measurement-based evaluation.

In Chapter 4, we introduce derivation-constrained access control and add support for it to the formal model and to the client-based access-control architecture. We also provide a measurement-based evaluation.

In Chapter 5, we investigate how confidential context-sensitive constraints can lead to information leaks in different access-control approaches. We incorporate constraints into the formal model and into the client-based access-control architecture. We also provide a measurement-based evaluation.

In Chapter 6, we introduce obscured proof-of-access descriptions and incorporate support for them into the client-based access-control architecture.

In Chapter 7, we present an encryption-based access-control architecture for pervasive computing and provide a measurement-based evaluation.

In Chapter 8, we conclude our work with a review of our contributions along with a discussion of some future research directions.

Chapter 2

Design of Access-Control Architecture

In this chapter, we introduce a distributed architecture for running access control in pervasive computing. Before we give the design of this architecture, let us put our research into context and compare it with related research directions.

2.1 Overview

Traditionally, access control has been studied in the context of *information security*. Information security rests on the confidentiality, integrity, and availability of information [20]. Confidentiality keeps information secret, integrity refers to its trustworthiness, and availability involves the ability to use information. In this thesis, we are concerned with the confidentiality of information.

Information security is not the same as *information privacy* [34], which is also a big concern in pervasive computing [80, 82]. In particular, privacy relates to personal control over one's personal information, whereas security relates to organizational control over information. For example, privacy involves people being in control of their information, people being given feedback about usage of their information, or long-term retention of their information [66]. Several research projects have started looking at these issues [66, 83, 8, 76]. Information security is required for achieving information privacy. If a service deployed by an organization did not employ security, the privacy of people that are part of or interact with this organization could be invaded.

In terms of information security, this thesis addresses confidentiality of information. There are three main research directions in this area [32], though the boundaries can be

fluent, especially in pervasive computing, as illustrated below. First, there is *access control*, which is what we are studying. Access control ensures that all accesses to information occur exclusively to the modes and rules fixed by protection policies. These policies state whether and how individuals in a system can access information offered by the system. For example, in the scenario introduced in Section 1.1, Alice can access Carol's calendar entry only if Carol grants this permission to Alice in a protection policy. Second, *flow control* regulates the flow of information in a system. In particular, information must not flow to an entity that is not authorized to access it. Carol's calendar entry, leaking her and Bob's location and activity information, can be looked at as a flow-control problem. Third, *inference control* tries to protect information from indirect detection. Namely, an entity should not be able to derive information that the entity cannot access directly by deriving the information (or at least some properties of the information) from information that the entity can access. Alice inferring Carol's current location based on the fate of her query for Carol's calendar entry can be looked at as an inference-control problem.

In this thesis, we use access-control techniques to avoid the flow-control and inference-control problems mentioned above. In particular, we will address information leaks caused by complex information in Chapter 3 and by confidential context-sensitive constraints in Chapter 5. We will elaborate on other techniques in the corresponding related-work sections. In this chapter, we present an access-control architecture for pervasive computing that is flexible enough to support the various access-control mechanisms presented in the reminder of this thesis.

Two main components of access control, and thus of an access-control architecture, are protection policies and enforcement of these policies. In this chapter, we first discuss protection policies for pervasive computing (Section 2.2). We then elaborate on different ways for the enforcement of such policies (Section 2.3). We discuss the application of these two concepts in the context of a people-location system (Section 2.4). The lessons learned during the design and deployment of this system contributed toward the design of protection policies and their enforcement in our access-control architecture for pervasive computing (Sections 2.5 and 2.6). We also state our security model (Section 2.7) and discuss related work (Section 2.8).

2.2 Access Policies

Access control requires protection policies ("access policies"). There are both mandatory policies and discretionary policies. In a mandatory policy, the organization running a service controls access to information and individuals cannot alter that access. For example,

in the Bell-LaPadula Model, information is classified (e.g., unclassified, confidential, secret, or top secret) and individuals are given corresponding clearances. The model prevents an individual from reading information that has a classification higher than the individual's clearance. In a discretionary policy, individuals can tell an access-control mechanism whether to allow or to deny access to information. Discretionary policies are based on the identification of the individual requesting access and the identity of the information.

In pervasive computing, individuals should be able to decide whom to grant access to their personal information. Therefore, pervasive computing calls for the decentralized administration of access policies, that is, discretionary policies.

2.2.1 Properties of Access Policies

In this section, we study the properties of access policies that we do or do not want to hold in the context of pervasive computing.

Distributed Access Policies

If Carol states that Alice can access her calendar and if Alice states Dave can also access Carol's calendar, then Dave should be able to access this calendar. Here, Carol's access policy is established by both Carol and Alice. Arguably, Carol might not want Alice to be able to grant access to Dave. However, even if she prevented Alice from doing so, Alice could still act as a proxy on Dave's behalf. (The question whether being granted access to information should imply being able to grant other people access has been discussed in the access-control research community for a long time.)

Therefore, multiple (authorized) entities should be able to establish an individual's access policy for personal information in pervasive computing. We call such access policies *distributed*.

Different Policy Issuers

We expect that in many scenarios, individuals (plus anyone to whom they grant access) will be responsible for defining access policies for their personal information. However, there are cases where a separate entity will specify access policies. For example, a parent might define access policies on behalf of a child. For a service run by an individual's employer, the employer might define the access policies for information about the individual provided by the service.

Therefore, access policies in pervasive computing must support the case where the entity establishing a policy for an individual's personal information is different from the individual.

Positive vs. Negative Access Policies

Typically, an access policy states that an individual can access a piece of information. However, it is also possible for an access policy to say that an individual cannot access a piece of information. For example, AFS [67] supports this feature. In this way, it becomes possible to grant access to all members of a group except some explicitly listed members. Consider the situation where Carol would like to grant all students in her class access to her calendar, except Alice. Therefore, she states in her access policy that class members can access her calendar. She also states that Alice cannot access her calendar. In order for Carol's policy to have the desired effect, access control must give her second statement precedence over the first one.

Denying access in an access policy raises several issues in terms of usefulness and enforcement of such policies. First, it is unclear how useful this feature is in practice. Second, being able to deny access makes the specification of access policies more difficult since it requires the issuer of a policy to understand the concept of precedence. Remember that in pervasive computing, access policies will often be defined by individuals, which might have only limited understanding of the mechanics of access control. Therefore, in this thesis, we strive to make access policies as simple as possible, while maintaining their usefulness. Third, policies that deny access are difficult to enforce in an environment lacking a centralized repository for access policies that is consulted by the entity making an access decision upon a request. Here, if access is denied in a policy, it will become difficult to ensure that all interested entities become aware of this denial. For example, in client-based access control, where a client needs to prove to a service that it is authorized to access the requested information, a client that is denied access in a policy has no interest in showing this statement to a service. Therefore, distributed access-control frameworks [21, 22, 47, 16, 69] typically do not support access denial in their policies. Formally, these frameworks require that an access policy regulating access to information is monotonic [22], that is, if an access policy consists of a set of statements and an individual is granted access based on a subset of these statements, then the individual is also granted access based on another subset of these statements, where the second subset is a superset of the first subset.

Due to these reasons, we do not support access policies that deny access in this thesis.

2.2.2 Access Rights

To satisfy the property of access policies being distributed, as outlined in Section 2.2.1, we build access policies out of *access rights*.

An access right is a self-contained statement, in which an entity states that another entity has access to a piece of information (assuming that the former entity has access in the first place). In this thesis, we will use digital certificates for expressing access rights, that is, an access right is a statement signed by its issuer. In this way, we can store access rights in a distributed way, detect tampering attacks on them, and identify the issuer of an access right. In addition, digital certificates are attractive since any individual (i.e., any owner of a public/private key) can issue digital certificates and hence access rights.

For an entity to be granted access to confidential information, there must be one or multiple access rights authorizing this access. Based on our discussion, an access right is specified as follows:

[**Issuer** grants **subject** access to **information** under some **constraints**.]

Let us elaborate on the various elements of an access right:

Issuers issue access rights.

Subjects are granted access rights. A subject can become an issuer by defining an access right and granting the access right to another subject. Therefore, there can be an entire chain of access rights that forward an access right. We call the first access right in the chain the *initial* access right.

The initial access rights to a piece of information plus any access rights forwarding an initial access right form a directed graph. This graph corresponds to the access policy of this information since it lists all subjects that can access the information and under what conditions. These access rights can be defined by multiple entities, and we achieve distributed access policies.

Information describes an information item (e.g., email or location) about a specific entity (e.g., Alice). For each type of information, there must be an *owner* who is allowed to define its initial access rights. For example, Alice defines the initial access rights for her location information. Depending on the scenario, the owner can be the individual the information is about, another individual (e.g., a parent), or the organization running the service providing the information. Associating information with its owner will allow us to easily support different policy issuers, as required in Section 2.2.1.

Constraints define under what conditions the access right is valid. There can be different types of constraints. For example, an access right is valid only during particular times of the day or for a certain location of the client that requests access to information. Similarly, a constraint can limit the lifetime of an access right. In derivation-constrained access control, a constraint requires that the information in the access right is used for answering a request for derived information, as discussed in Chapter 4.

In order for a client requesting information to be able to exploit an access right to this information, the access right must be valid, that is, any constraints attached to the access right must be satisfied and the subject of the access right must correspond to the client. In addition, the issuer of the access right must either own the information in the access right or be the subject of another valid access right to the same information that is exploitable by this issuer.

Either the subject or the constraints can be omitted from an access right. Leaving the subject out of an access right specification is a shortcut for granting the access right to all possible subjects. If the constraints are omitted from an access right, they will not be taken into account for ensuring its validity.

It is possible to add additional elements to an access policy. For example, in the context of enterprises dealing with customer privacy, *privacy policies* have been introduced [8, 76]. A privacy policy includes an access policy as discussed in this section. In addition, it also contains a *purpose* part (what the information can be used for (e.g., marketing)), an *action* part (what actions can be taken on the information (e.g., read or write)), and an *obligations* part (what duties must be fulfilled when using the information (e.g., notify owner of information)). In this thesis, we concentrate on access policies; they are a required component for implementing privacy policies.

As mentioned in Section 2.2, discretionary access control requires authentication of a client. This authentication can be arbitrarily complex. For example, if an access right is granted to Alice and somebody requests access by presenting Alice's alleged public key, the mapping from Alice to this public key must be validated. This mapping process has been well investigated. For example, it can be solved with a PKI together with X.509 certificates. Therefore, we avoid the mapping in this thesis and assume that both the subject and the issuer of an access right correspond to a public key. In addition, our discussion will treat public keys and their owners equivalently. For instance, we do not differentiate between Alice and Alice's public key. In this simplified model, authenticating a client corresponds to ensuring that the client owns the private key corresponding to its alleged public key. Again, this authentication has been well researched (e.g., SSL with

client authentication).

2.3 Access-Control Architectures

After having examined access policies for pervasive computing, we now discuss four architectures for the enforcement of these policies. In particular, we study centralized, service-based, client-based, and end-to-end access control.

2.3.1 Centralized Access Control

In centralized access control, a client sends a request for information to a centralized entity, which runs access control on behalf of individual services. If there are access rights that grant access to the client, the centralized entity will retrieve the requested information from a service and return the information to the client. The services in an environment trust the centralized entity to properly run access control on their behalf and are willing to give any information to the centralized entity. Many pervasive computing frameworks that support access control pursue a centralized approach [5, 38, 42, 96, 132, 121, 75].

The advantage of centralized access control is its flexibility. First, since the centralized entity has access to any information, all this information can be included when making an access decision. Second, it is reasonable to assume that the entity has sufficient computing resources and that it can use a powerful rule engine to reason about an access decision.

Centralized access control has the drawback of a single point of failure. Since every service trusts the centralized entity, an intruder into this entity will get access both to the knowledge stored in the entity and to the information provided by all the services in the environment. In addition, since every request has to go through this entity, the entity can become a performance bottleneck. Furthermore, it is unclear how the approach scales to multiple pervasive computing environments. Will there be a single entity shared by all the environments? Will there be a centralized entity per environment? The former approach does not look realistic. The latter approach raises consistency issues: How can we ensure that each environment uses the same knowledge when making an access decision? Finally, while a centralized approach allows for flexible access control, the flexibility can make it undecidable for the centralized entity to decide whether a request should be granted access [81, 7].

2.3.2 Service-based Access Control

In service-based access control, a client contacts a service directly. Each service controls access to the information that it provides. If there are access rights granting access to a client, a service will return the requested information to the client.

Service-based access control avoids a single point of failure. An intruder breaking into a service has access only to information provided by this service (and information to which the service has access), but not to all information provided by, for example, all the services in an environment.

The drawback of this approach is its potentially limited flexibility. A service might not have access to all the information provided by the services in an environment and thus cannot exploit this information for making an access decision. In addition, making such a decision can be expensive for a service, especially if this task involves contacting other services in order to locate access rights or to resolve constraints on access rights. Finally, this approach also suffers from the drawback that access control can become undecidable.

We can reduce the load on a service due to access control by assigning some of the load to a client or to other services. We investigate these two approaches in the next two sections.

2.3.3 Client-Based Access Control

In client-based access control, a client contacts a service directly. The client needs to prove to the service that it is authorized to access the requested information. A proof of access consists of one or multiple access rights. The service makes the access decision by ensuring the validity of the proof submitted by the client. This approach is less expensive for a service since it shields the service from performing potentially expensive tasks, such as locating required access rights or resolving constraints. Validating an access right tends to be cheap. In existing client-based access-control frameworks [16, 69], access rights are expressed as digital certificates in order to ensure their integrity.

An advantage of client-based access control is that the validity of a proof of access can be checked even if the proof is based on an undecidable logic [7]. In addition, building a proof of access can be easier for a client than for a service since it can be done in a stepwise fashion [81]. Alternatively, a client can limit itself to a decidable subset of the logic [16]. Another advantage of this approach is that there is no single point of failure. Similar to service-based access control, an intruder into a service gets access only to information provided by this service (and information to which the service has access).

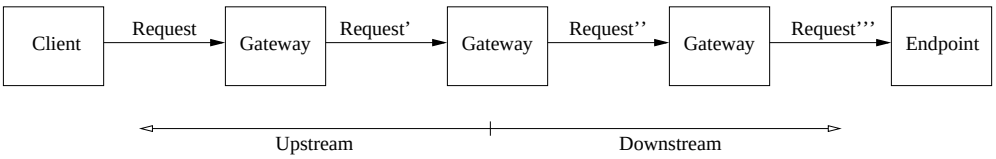


Figure 2.1: Information Gateways. Services are either gateways or endpoints. A request sent to a gateway by a client can result in further requests to other gateways, and so on, till a request reaches an endpoint.

Due to these advantages, we focus on a client-based approach for the access-control architecture presented in this thesis. It is possible for computationally weak clients to offload proof building to a proxy. Furthermore, our architecture supports the usage of powerful, potentially centralized rule engines for building proofs of access, which enables more flexible access control. However, we envision that most requests for information require only simple proofs of access, which can be dealt with in a fully distributed fashion by our architecture.

2.3.4 End-to-End Access Control

As shown in the scenario in Section 1.1, services might have to communicate with other services in order to answer a client's request for information. For example, a location service exploiting the location of a person's laptop for locating the person might have to contact access points. Similarly, a service providing availability information about a person might have to contact a service providing location information about this person. This service in turn might contact other services. In the rest of this thesis, we call services that issue further requests *gateways* and services that do not *endpoints*, as illustrated in Figure 2.1. It is possible that a gateway contacts multiple services.

Service-based (and client-based) access control requires each service to make an access decision. When we extend this model to a world with gateways, each gateway and endpoint needs to make an access decision. A service's decision is based either on the identity of the client requesting information or on the identity of the gateway requesting information. In the latter case, the decision takes neither the identity of the client nor the identity of any gateways upstream of the requesting gateway into account. We call this model *step-by-step access control*, since access control for each step is independent of the previous steps.

Alternatively, to reduce the load on gateways, it is possible to have only the endpoint(s) reached by a series of requests make an access decision, but not any of the gateways. In this

model, an endpoint's decision is based on both the identity of the client and the identity of all the upstream gateways. We call this model *end-to-end access control*. The model still requires gateways to cooperate. Namely, a gateway needs to inform the next downstream gateway or the endpoint of the identity of the client and of any upstream gateways, so that the endpoint can consider these entities in its access decision. For example, Howell and Kotz [69] exploit end-to-end access control.

End-to-end access control has the advantage that it reduces the load on some of the services since gateways are no longer required to make an access decision. However, it can increase the load on an endpoint. For example, in client-based access control, a proof of access covering a gateway and a client can be more expensive to validate than a proof of access covering only the client or only the gateway (see Chapter 4). Therefore, in terms of overall access-control load, neither of the two approaches is a clear winner; the right choice depends on the deployment environment. If an endpoint was a computationally limited device, step-by-step access control looks attractive because of its reduced load on endpoints. (However, recent advances are making this limitation disappear [57].) Similarly, if a gateway was a simple device, such as a filter that reduces the granularity of location information, and is not able to make an access decision, the end-to-end approach might be better suited.

A significant advantage of end-to-end access control is that it allows us to limit the power of access rights granted to a gateway. As explained in Section 1.1.3, services that derive information and that have access to raw information become attractive targets for attackers. With the help of end-to-end access control, we can limit the influence of an attacker into a gateway by making gateways prove to endpoints that they are asking for information on behalf an authorized client. We elaborate on this approach in Chapter 4.

For simplicity reasons, we concentrate on the interaction between a client and the first gateway in most of this thesis. However, all the presented access-control mechanisms can also be applied to interactions between services. Similarly, we assume step-by-step access control and have the first gateway make access decisions. Again, it is possible to incorporate our presented mechanisms into end-to-end access control. We will explicitly address end-to-end access control and interactions between services in Chapter 4.

After having reviewed four access-control architectures, we now present two sample implementations in the next sections.

2.4 People-Location System

Before building a client-based access-control architecture for pervasive computing, we designed and deployed a distributed architecture for a subset of the information offered in pervasive computing, namely, for location information [61]. In this section, we review this system, present the access rights that we use for implementing access policies, explain how the system distributes access-control load, and discuss several lessons that affected the design of the more general access-control architecture.

2.4.1 Overview

We assume that the location system has a hierarchical structure. The system consists of multiple location services. Each service either exploits a particular technology for gathering location information or processes location information received from other location services. Figure 2.2 shows an example of such a system, as it could be deployed in Carnegie Mellon's School of Computer Science (SCS). A client contacts the People Locator service at the root of the system. This service then contacts other services, which themselves may also contact other services. Location information flows in the reverse direction of a request (not shown in the figure).

There are two groups of location services. The first group consists of services that are aware of the location of people. The second group includes services that are aware of the location of devices. These services can locate a user indirectly by locating the device(s) that the user is carrying with her. The People Locator service, the Calendar service, and the Device Locator service belong to the first group. The People Locator service aggregates location information received from other services. The Calendar service looks at people's appointments to determine their current location. The Device Locator service maps a query for a person to potentially several queries for her devices and contacts services in the second group. In our example, this group of services consists of the Wi-Fi service and the GPS service. The Wi-Fi service keeps track of the location of wireless devices by identifying the access point(s) they are connecting to. Different organizations may administrate the various services.

2.4.2 Access Rights

A location service grants access to clients based on access rights. An access right can restrict the granularity of the returned location information. In addition, it is possible to

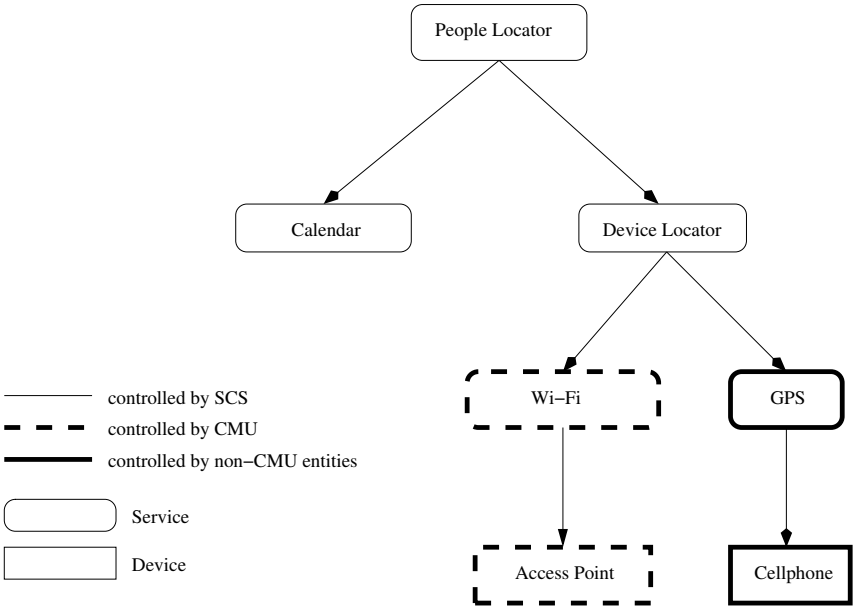


Figure 2.2: People-location system. The People Locator service forwards a query for the location of a person to the Calendar service and to the Device Locator service. The Device Locator service locates a person by locating her devices. It queries the Wi-Fi service for the location of the person's laptop and the GPS service for the location of the person's cellphone.

```
(cert
   (issuer (pub_key:people_locator))
   (subject (pub_key:alice))
   (propagate)
   (tag (policy alice)))


(cert
   (issuer (pub_key:alice))
   (subject (pub_key:bob))
   (propagate)
   (tag (policy alice
        (* set (* prefix world.cmu.wean) world.cmu.doherty.room1234)
        (* set (monday (* range numeric ge #8000# le #1200#))
          (tuesday (* range numeric ge #1300# le #1400#)))
        coarse-grained)))
```

Figure 2.3: SPKI/SDSI certificate. We show an access right in which the People Locator service lets Alice decide about her location policy and an access right in which Alice grants access to Bob. `pub_key:foo` stands for foo's public key. (Digital signatures are omitted.)

indicate a set of location and time intervals. A service will return location information only if the queried individual is at one of the listed locations and during these time intervals, respectively.

We use SPKI/SDSI digital certificates [47] for expressing access rights. Authentication and authorization based on SPKI/SDSI certificates do not rely on the existence of a global naming structure as, for example, the one introduced for X.509 certificates. Instead, the authentication and authorization step are merged, and a certificate gives access rights directly to a public key. Tools exist that evaluate chains of SPKI/SDSI certificates (i.e., forwarded access rights) and decide whether requests should be granted access [69].

SPKI/SDSI certificates are encoded as S-expressions, which are Lisp-like expressions. We give two example certificates in Figure 2.3. Keywords are printed in bold. In the first certificate, the People Locator service grants Alice access to her location information. The **issuer** entry corresponds to the issuer of the access right. As discussed in Section 2.5, we assume that this issuer is a public key. The **subject** entry lists the subject of the access right. Similar to the issuer, it is a public key. The keyword **propagate** states that forwarding is allowed. Alice can thus issue additional certificates that grant other people access to her location information. The entries following the keyword **tag** specify the type of

the certificate (e.g., `policy` for a location policy) and its scope (e.g., `alice` for Alice's location policy).

In the second certificate, Alice grants Bob access to her location information. Bob can locate Alice only if she is either in Wean Hall or in Room 1234 in Doherty Hall and on Monday between 8am and 12pm and on Tuesday between 1pm and 2pm. Finally, Bob has only coarse-grained access to Alice's location information.

2.4.3 Distributed Access Control

In the people-location system, when a client sends a request to a service, the client also submits access rights that show that the client is authorized to access the requested information. For example, when contacting the People Locator service, Bob needs to submit the two certificates shown in Figure 2.3 in order to be granted access to Alice's location information. We also employ the concepts of *service trust* and *delegation* to reduce the influence of an attacker into a service and to avoid redundant access-control checks, respectively. Let us discuss these two concepts in more detail.

Service Trust

Some of the services in the location system, such as the People Locator service shown in Figure 2.2, do not generate their own location information. Instead, they forward received requests to other services and process the answers from these services. To avoid information leaks, the location system must ensure that only services that implement access control are given location information by other services. One way to implement this condition is to require that a service has an access right to the location information of the queried individual. Having an access right means that a service can actively issue requests for information. This option is appropriate for services that must be able to issue requests for location information. For example, the Device Locator service shown in Figure 2.2 has to generate location queries for devices upon receiving a location query for an individual. However, for services such as the People Locator service, this option gives a service more privileges than the service really needs to have and violates the principle of least privilege [108]. Namely, by granting it the right to issue requests, we increase exposure in case of a break-in. If an intruder issues requests, the location system will grant access to these requests.

Due to these reasons, we introduce the concept of *service trust*. If a service that does not have an access right (i.e., it cannot actively issue requests) is trusted, it is given location

information when forwarding an authorized request from someone else. For a trusted service, the trust assumption is that the service implements access control in the following way:

- The service ensures that there is an access right for the entity that issued a request. If this check fails, access will be denied.
- The service then checks whether the entity from which it received the request corresponds to the entity that issued the request. If it does, access will be granted. If it does not, the service must have received a forwarded request. Therefore, the service has to ensure that the entity from which the service received the request is trusted. If so, access will be granted, otherwise access will be denied.

Service trust is a hybrid between step-by-step and end-to-end access control, as defined in Section 2.3.4. There is end-to-end access control since an endpoint can take both the identity of a client and of a gateway into account when making an access decision. There is step-by-step access control since a gateway also makes access decisions. End-to-end access control reduces the influence of an attacker into a gateway. Step-by-step access control limits the impact of a denial-of-service attack, since unauthorized requests are thrown away by a gateway already and do not reach an endpoint.

Similar to access rights, we use SPKI/SDSI certificates for defining trusted services. The entry in the tag section needs to be modified accordingly. For example, `(`**tag** `(trust alice))` identifies certificates that declare Alice's set of trusted services.

Delegation

Each location service has to implement access control, as defined in the previous section. In particular, when receiving a set of access rights with a request, a service needs to ensure that these rights grant access to the issuer of the request. However, to reduce overhead or in the case of low processing power, we do not require each service to validate the potentially many access rights itself. Instead, a service can delegate this task to another service. For example, the Calendar service shown in Figure 2.2 is likely to delegate access control to the People Locator service since both services are run by the same organization and the People Locator service also needs to validate the access rights. After this validation, the People Locator service issues an access right (in a certificate) that directly authorizes the issuer of the request. The service gives this access right to the Calendar service when forwarding the request. Now the Calendar service has to validate only this access rights for its access decision, not all the access rights underlying this access right.

2.4.4 Lessons

Several lessons learned during the design and implementation of the people-location system contributed toward the design of the access-control architecture for information in pervasive computing described later in this chapter. We now discuss these lessons.

Information-representation scheme. In the people-location system, we rely on a simple scheme for representing the information to which an access right grants access. In particular, as shown in Section 2.4.2, we use the userid of an individual. This scheme is not powerful enough. For instance, the scheme has no support for information other than location information; the implicit assumption is that access rights grant access to location information. Therefore, an attack is possible where an individual exploits an access right to location information for accessing other information that happens to use the same representation scheme in its access rights. Moreover, the scheme assumes that all location services assign the same userid to a particular individual, which is not a realistic assumption when these services are run by different entities. These drawbacks lead us to the definition of the information-representation scheme introduced in Section 2.5.1 for the more general architecture.

Information and constraints. We employ SPKI/SDSI certificates for expressing access rights. Our notion of an access right given in Section 2.2.2 distinguishes between the information to which access is granted and the constraints imposed on accessing this information. SPKI/SDSI certificates do not support this separation. Their design assumes that the **tag** entry of a certificate lists both the information and any constraints. When combining two certificates, such as the ones shown in Figure 2.3, SPKI/SDSI's default combination mechanism performs only simple string comparison operations between the two certificates and does not consider the nature of the contents in the **tag** entry. This mechanism makes it difficult to, for example, exploit the semantics of information for access control. Therefore, for the more general architecture, we add a new entry to SPKI/SDSI certificates, which lists the information to which access is granted (see Section 2.5.3). In Chapter 3, we will exploit this separation to take the semantics of information into account for access control.

Service trust. The concept of service trust helps only against intrusions into services that forward requests, but not against intrusions into services that need to issue new requests. For example, the Device Locator service shown in Figure 2.2 does not benefit from service trust in case of an intrusion, since it needs to issue a new request for the location of Alice's laptop. Another problem with service trust is that it is a very coarse-grained notion. A service is either trusted or not trusted. For example,

it is not possible to specify what actions a service is trusted to perform for what information. In Chapter 4, we address these two issues by formalizing derivation properties of information.

Delegation. Delegation of access control aims to avoid redundant validation of access rights by having the first service that performs this validation issue a summary access right. However, delegation does not necessarily lead to a decrease in the response time experienced by a client. In particular, for some asymmetric cryptographic algorithms, such as RSA, issuing a certificate representing an access right is much more expensive than validating such a certificate. Therefore, delegation pays off only if many access rights would have to be re-validated by the second service. In general, checking the validity of access rights is a cheap operation, when compared with the cost of other operations required for answering a client's request (e.g., setting up a secure connection or retrieving the actual location information [61]). Therefore, in the more general access-control architecture, we do not support delegation of access control.

Separation between people and devices. In the people-location system shown in Figure 2.2, we separate between services that locate people and services that locate devices. This separation is based on the observation that in the latter case, we must map a person to a device and then locate this device. However, this separation is artificial. We also need such a mapping for a service that locates people. For example, for a service that exploits a calendar for locating a person, we need to map the person to a file or to a userid used by the calendar application. For a service that detects a person's face in a videostream, we must map the person to a set of biometric parameters. Therefore, in the more general architecture, we no longer separate between people and device locator services.

Proofs of access. In client-based access control, a client submits a proof of access, which shows that the client is authorized to access the requested information, to a service. A proof of access contains the access rights required by the client and instructions that tell the service how to combine these access rights in order to conclude that the client should be granted access. In the people-location system, a client submits only a set of access rights, but no instructions. We chose this solution for simplicity reasons. However, once we incorporate the semantics of information into access control, proofs of access will contain not only access rights, but also information relationships (see Chapter 3). Here, it becomes essential that a client tells a service how to combine access rights and information relationships. Therefore, in the more general architecture, we have a client transmit an actual proof of access.

Based on the lessons that we learned from the people-location system, we now present a more general access-control architecture. We first examine what access rights look like in this architecture (Section 2.5). Then, we describe how we achieve distributed access control (Section 2.6).

2.5 Access Rights

The discussion in Section 2.4.4 revealed two weaknesses of access rights in the people-location system. First, the information-representation scheme is insufficient. Second, when assembling access rights in a proof of access, we also need to include instructions on how to combine these access rights. In this section, we introduce the information-representation scheme used in our access-control architecture for pervasive computing. We also present a formal model of access rights and their combination. Finally, we describe the digital certificates that we employ for expressing access rights.

2.5.1 Information Representation

We need a representation scheme for the information that an access right grants access to. We have the following two requirements for this representation:

Service and environment independent. The representation scheme should not bind information to a service that offers this information. For example, URLs, which are used for identifying information on the Web, do not fulfill this requirement. If information was bound to a particular service, an access right to this information would also be bound to this service. When multiple services offer the same information, as it can be the case in pervasive computing, each service requires its own access rights. This increases the number of access rights that the owner of the information needs to define and thus raises the danger of mistakes and information leaks. The same reasoning calls for an information-representation scheme that is independent of the environment in which an access right is used. Having an information-representation scheme that is service and environment-independent is a common approach for pervasive computing frameworks that address access control to information [66, 39, 51].

Owner of information. Discretionary access control lets the owner of information issue initial access rights to this information. In order to differentiate between valid and

invalid access rights, access control must be able to learn about the owner of information for which an access decision has to be made. This should be straightforward, which suggests that the owner becomes part of the representation scheme for information.

These two requirements lead us to represent information in the following way:

$$(\mathbf{owner}, \mathbf{item}).\mathbf{type}.$$

Let us elaborate on the various elements of this scheme:

Owner. The owner is responsible for issuing initial access rights to the information described by the scheme. For example, if Bob owns his location information, he will issue initial access rights to his location information.

Item. Information is about a particular item (e.g., Bob, his car, a room in his house, or an event that he organizes).

Type. Information is of a particular type (e.g., location, activity, financial, or temperature).

Each component of the representation should be associated with a formal description of the component. For example, the description could say that a public key representing a particular owner is an RSA public key of 1024 bits. Similarly, it could say that item Bob is a person or it could define the type "location". With the help of this description, a service offering information can ensure that its notion of the information corresponds to the owner's understanding, as expressed in an initial access right to this information. Similarly, a client that exploits an access right to information can ensure that its notion matches the owner's understanding.

For the formal description of information, we can exploit descriptions developed for the Semantic Web, based on OWL [45]. These *ontologies* make it possible to identify, for example, different types of information that have the same semantic meaning (e.g., "location" and "whereabouts") or that are expressed in different languages (e.g., "location" and "Standort"). Using this approach, the description of a component in the information-representation scheme consists of a URL pointing to an ontology. For example, SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) [39] contains both an OWL representation of "person" (`http://pervasive.semanticweb.org/ont/2004/06/person`) and of "location" (`http://pervasive.semanticweb.org/ont/2004/06/location`).

With the help of ontologies, we can ensure that different individuals or services agree on the meaning of a component in the information-representation scheme. For example, they agree that a particular item is a person, but not a car. However, different individuals or services could still use different schemes for naming the same owner or item. For example, a service could use "(alice, alice).location" when referring to Alice's location, where "alice" is Alice's userid, as used by the service. On the other hand, when issuing an initial access right to her location information, Alice refers to "(⟨Alice's public key⟩, Alice).location". (We use ⟨⟩ as a placeholder for the actual public key.) Related to this observation, we point out that the string denoting the item has meaning only in the namespace of the owner of the information. For example, the room that building manager Alice identifies as "Wean Hall 8220" might not correspond to the room that building manager Bob identifies as "Wean Hall 8220". Local namespaces have been used in previous research [47, 87]. They rely on the observation that the existence of a global namespace, which could avoid this confusion, is unrealistic.

To cope with different naming schemes for the same owner or item, a service must ensure that for information offered by the service, the owner and the item really correspond to the owner and the item listed in access rights that the service is going to use when controlling access to this information. This validation can occur manually or automatically. Manual validation can take place when the issuer of an initial access right registers with the service. For automatic validation, there could be ontologies that relate different ownership representations and different local namespaces. (Formally, we could also use bundling-based information relationships, as presented in Chapter 3, for automatic validation.)

Given the existence of such mappings between owners or items, we assume that all individuals and services agree on the representation and naming scheme for information. In addition, for simplicity reasons, we assume that the owner of information is represented as a public key. Using our representation scheme, we can describe Alice's location as follows:

(⟨Alice's public key⟩, Alice).location.

Similarly, we can describe the temperature in Wean Hall 8220 as

(⟨Public key of building manager⟩, Wean_Hall_8220).temperature.

As mentioned in Section 2.2.2, we do not separate between public keys and their owners. Therefore, Alice's location information becomes

(Alice, Alice).location.

We use the shortcut

$$\mathbf{owner.type}$$

for representing information where the owner and the item are identical. Using this shortcut, Alice's location is represented as

Alice.location.

As outlined in Section 2.2.1, Alice's location information is not necessarily owned by Alice. For example, in an enterprise environment, this information could be owned by the enterprise. In such a scenario, we can represent Alice's location as follows:

(Enterprise, Alice).location.

In general, a service (i.e., its administrator) decides about ownership of information provided by the service.

We use this information-representation scheme for describing the information that an access right grants access to. In Chapter 3, we will show how we can exploit the same representation scheme in information relationships in order to capture the semantics of information for access control.

2.5.2 Formal Model

In this section, we present a formal model that defines how access rights can be combined in proofs of access. Here, we restrict ourselves to access rights that do not include any constraints. Later, we will extend the formal model to support more complex proofs of access that take the semantics of information into account (Chapter 3), make gateways more secure in case of an intrusion (Chapter 4), or enable access rights with constraints (Chapter 5).

In a distributed environment, multiple entities issue statements, such as access rights to information or requests for information. Some entities might be malicious, so not all of these statements will be valid. During access control, it must be possible to authenticate valid statements and to ignore statements that are not relevant. Lampson et al. [81]

introduce a theory of authentication, which formalizes authentication in distributed environments. In this section, we review this theory and discuss how we use it for dealing with access rights in proofs of access.

The "speaks-for" Relationship

Lampson et al.'s theory of authentication consists of principals that make statements. Principals are either simple or compound. There are two types of simple principals: named principals and channels. Named principals can be people, machines, roles, or sets of principals. Channels are principals that can say things directly, such as a wire, an encrypted channel, or a network address. Only channels can directly make statements to a computer. A person, for example, must communicate over a channel to a computer. Compound principals are built up out of other principals. For example, compound principals allow principals to act in a particular role. In this thesis, we concentrate on simple principals (except in derivation-constrained access control in Chapter 4). However, the presented concepts can also be applied scenarios with compound principals.

For expressing access rights to information, we exploit Howell and Kotz's "restricted speaks-for" relationship [68], which is based on Lampson et al.'s "speaks-for" relationship. We present the relationship based on an example statement. The statement $B \stackrel{T}{\Longrightarrow} A$ denotes that principal B speaks for principal A regarding the statements in set T, that is, if B issues a statement that is in T, A also issues this statement, or

$$\vdash (B \stackrel{T}{\Longrightarrow} A) \supset ((B \textbf{ says } s) \supset (A \textbf{ says } s)) \text{ with } s \in T. \tag{2.1}$$

($\supset$ denotes implication.)

We are interested in statements expressing read access to some information, for example, "read $C.x$" or "read $(C, D).x$" using the representation scheme introduced in Section 2.5.1. In terms of our formal model, C and D are (simple) principals. In particular, C denotes the principal that owns the information and D is the principal the information is about. D does not have to be a person, it can also be, for example, a device or a room. In the rest of this paper, we use the shortcut $B \stackrel{C.x}{\Longrightarrow} A$ instead of $B \stackrel{\{\text{"read } C.x\text{"}\}}{\Longrightarrow} A$. As we will see below, this speaks-for relationship grants B access to information $C.x$, assuming that A itself has access to this information.

Access Rights

Speaks-for relationships are derived from access rights. Lampson et al.'s "handoff axiom" says that such a relationship holds only if it is made by the principal on the right-hand side of the statement. Formally,

$$\vdash (A \textbf{ says } (B \stackrel{C.x}{\Longrightarrow} A)) \supset (B \stackrel{C.x}{\Longrightarrow} A). \tag{2.2}$$

A statement of the form $A \textbf{ says } (B \stackrel{C.x}{\Longrightarrow} A)$ roughly corresponds to an access right introduced in Section 2.5. (The statement has no support for constraints.) For example, the access right "Alice grants Bob access to Carol's location" is expressed in the formal model as $\mathrm{Alice} \textbf{ says } (\mathrm{Bob} \stackrel{\mathrm{Carol.location}}{\Longrightarrow} \mathrm{Alice})$. An access right that does not include a subject can be represented in the formal model as a set of such statements, one for each possible principal.

A useful property of the speaks-for relationship is transitivity, or

$$\vdash (D \stackrel{C.x}{\Longrightarrow} B) \wedge (B \stackrel{C.x}{\Longrightarrow} A) \supset (D \stackrel{C.x}{\Longrightarrow} A). \tag{2.3}$$

Transitivity, together with the handoff axiom, allows for the forwarding of access rights and defines how clients can combine access rights in a proof of access. We will explore how a service formally exploits such a proof for its access decision in Section 2.6.2.

2.5.3 Implementation

Similar to the people-location system, we express access rights as SPKI/SDSI digital certificates [47]. As mentioned in Section 2.4.4, the existing definition of SPKI/SDSI certificates is not sufficiently powerful for our purposes, since it merges the information to which access is granted and the constraints on an access right in a single entry. Therefore, we modify the format slightly and introduce a separate entry for the information to which access is granted, which allows us to keep this information separate from the imposed constraints.

We give a sample certificate in Figure 2.4. The **version** entry indicates that we are using non-standard SPKI/SDSI certificates. The **issuer** and **subject** entries list the issuer and the subject, respectively. The **permission** entry is our extension to SPKI/SDSI certificates;

```
(cert
   (version ''1'')
   (issuer (pub_key:alice))
   (subject (pub_key:bob))
   (permission
      (information (pub_key:alice) alice location))
   (tag *))
```

Figure 2.4: Extended SPKI/SDSI certificate. We show an access right in which Alice grants Bob access to her location information. `pub_key:foo` stands for foo's public key. (Digital signatures are omitted.)

it describes the resource to which access is granted. Currently, we allow only resources of type **information**. It is possible to add support for other resources, such as physical resources (e.g., a projector). As discussed in Section 2.5.1, the information-representation scheme consists of three parts: the owner of the information (which is a public key), the item the information is about, and the type of information. The **tag** entry lists any constraints imposed on the access right. We will take a closer look at constraints in Chapter 5.

2.6 Distributed Access Control

Our access-control architecture is client-based, meaning that a client has to prove to a service that it is authorized to access the requested information. There are three main components in this architecture: proof building, access rights storage, and proof verification. In terms of proof building by a client, we presented the underlying formal model in Section 2.5.2. In this section, we discuss the storage of the access rights used in proof building and the formal validation of a proof of access by a service. In addition, we review the implementation and deployment of our architecture.

2.6.1 Access-Rights Storage

In our architecture, we assume that a client that has been granted access to information stores the corresponding access rights locally. As mentioned in Section 1.4, we do not examine remote certificate discovery in this thesis. Instead, we assume that the issuer of an access right hands over the corresponding certificate to the subject of the access right,

together with any access rights showing that the issuer also has access to the information.

2.6.2 Proof Verification

In Section 2.5.2, we presented a formal model for the composition of access rights in a proof of access. We now discuss how a service makes an access decision based on proofs of access represented in this formal model.

When a service receives a request "read $D.x$" from principal C, it needs to ensure that C speaks for a principal on the service's access control list ("ACL") for this particular resource. Abadi et al. [1] introduce the construct D **controls** t to encode that principal D is on a service's ACL for resource t. The construct is defined as

$$D \textbf{ controls } t \equiv ((D \textbf{ says } t) \supset t).$$

As mentioned in Section 2.5.1, owner D of information $D.x$ is responsible for defining the initial access right to $D.x$, formally, D **controls** "read $D.x$".

When C requests $D.x$ from a service (i.e., C **says** "read $D.x$"), C also needs to submit a proof of access composed of access rights showing $C \stackrel{D.x}{\Longrightarrow} D$ to the service. Given these two statements and the service's ACL (i.e., D **controls** "read $D.x$"), the service concludes "read $D.x$" and grants C access to $D.x$.

In our scenario, C is the channel over which the service receives a request. We assume that channels are authenticated (e.g., an SSL connection with client authentication), which lets them speak for a principal, B, at the other end of the channel, or $C \Rightarrow B$. Principal B can be a public key. This public key, in turn, can speak for the person owning this public key. In the rest of this thesis, we leave channels and people away for simplicity reasons. We will assume that principals are public keys and that they can speak directly to a service.

2.6.3 Implementation

The implementation of our access-control architecture is based on Howell and Kotz's framework for client-based access control in Web environments [69]. At the beginning of the thesis research, this framework was the only implemented client-based access-control architecture with publicly available source code that provided facilities for storage of access rights and building and verification of proofs of access. Another advantage was that

the framework expresses access rights as SPKI/SDSI certificates, which we had already used for the people-location system.

In the access-control framework, proofs of access are implemented as Java classes. Each axiom in the formal model has its corresponding class. For example, there is a class for the transitivity axiom (Axiom (2.3)). Each instance of a class is initialized with the preconditions of an axiom. The initialization values can be instances of proof classes themselves. Furthermore, each class has a verification method, which ensures that the preconditions can be combined in the formal way defined by the axiom underlying the class. This method is called by a service. A class is able to serialize and deserialize its instance variables. Similar to SPKI/SDSI certificates, a serialized proof of access is encoded as an S-expression. A service must instantiate only proof classes representing axioms from our formal model when deserializing a proof of access.

We extended Howell and Kotz's framework to support the information-representation scheme introduced in Section 2.5.1 and the modified SPKI/SDSI certificates presented in Section 2.5.3. Later, we will add additional access-control mechanisms to this architecture.

2.6.4 Deployment

We use the Contextual Service Interface (CSInt) [73] developed for the Aura pervasive computing project [52] as a testbed for the deployment of our access-control architecture. CSInt is client/server-based and allows a client to retrieve information from a service using a simplified SQL query language. It is implemented in Java, exploits HTTP requests and responses for issuing queries and receiving results, respectively, and encodes SQL queries and results for these queries in XML. To enable support for client-based access control, we modified the CSInt as follows:

- In addition to transmitting the various components of an SQL query (i.e., selected attributes, name of database or table, and where clause) and requirements for the answer to this query (e.g., desired freshness or confidence), we have the CSInt transmit a proof of access, if desired by a client. Since proofs of access are serialized into S-expressions, we enhanced the CSInt to convert an S-expression into an XML datastructure (and vice-versa) and to transmit this datastructure. The CSInt is oblivious to the type of S-expression.

- To achieve confidentiality and integrity of a query and its response, we extended the CSInt to exploit SSL as its underlying transport protocol. In particular, we use a modified version of Claymore PureTLS [119]. The modifications allow for the usage

of SPKI/SDSI certificates, instead of X.509 certificates. SSL also provides client authentication, which is required for discretionary access control. An additional benefit is server authentication to avoid man-in-the-middle attacks.

2.7 Security Model

When building a security mechanism, it is advisable to have a model of the system in which the mechanism is deployed and of the capabilities of attackers targeting this mechanism. In this section, we describe the security model underlying our access-control architecture for pervasive computing.

In our system, there are services that provide confidential information and clients that access this information. A service can become a client itself. Different administrative entities control these services. Therefore, before handing confidential information over to a client, a service must ensure that the client is authorized to access this information. Namely, a service employs the access-control algorithms described in this thesis.

The offered information can be complex, that is, it reveals other types of information. Furthermore, access decisions can be constrained based on confidential information about an individual's context.

The goal of an attacker is to learn confidential information that the attacker is not authorized to access. The attacker is not interested in other malicious behavior, such as modifying information returned by a service. (Though we want to make such behavior harder by, e.g., using authenticated communication.)

In order to achieve this goal, an attacker can choose between the following actions: An attacker can send queries to a service and observe their fate. A query is either denied or granted access. In the latter case, the attacker will see the information that the query asked for. Alternatively, an attacker can set up its own services and observe queries reaching such a service. The attacker can also snoop network traffic. Attackers can collaborate.

In this thesis, we do not address the following attacks:

Service-specific attacks allow an attacker to infer confidential information based on knowledge about the properties of a particular service. For example, if the response time for learning an individual's location depends on the location of the individual, timing attacks will become possible. If an individual's location is derived from the location of her device, attacks based on individuals swapping devices among each other will be feasible.

Information-specific attacks enable an attacker to infer confidential information from the current value of information that the attacker is authorized to access. For example, for some locations (e.g., a gas station), it is easy to deduce the activity of an individual being at one of these locations (or at least a small set of possible activities).

Traffic-analysis attacks let an attacker deduce confidential information by observing which services exchange traffic with each other. For example, if an attacker knows that calendar information about an individual is accessible only for particular locations of the individual, the (non)occurrence of queries between the location and calendar services and a client might allow the attacker to learn the individual's location.

Statistical-inference attacks let an attacker send multiple queries to (different) services and derive confidential information that the attacker is not authorized to access. For example, if Alice learns that Bob and Carol are at the same location by querying a location service for Bob and Carol's location and that Carol is in a meeting by querying an activity service for Carol's activity, Alice will also figure out Bob's activity, though she might not be authorized to access this information.

Service-intrusion attacks allow an attacker to learn confidential information by breaking into a service offering this information. We ignore this type of attacks in most of the thesis, except in Chapter 4, where we make services more secure in case of an intrusion. We elaborate on the necessary changes to the security model in Section 4.1.

Outside attacks let an attacker access confidential information by exploiting channels that our system is not aware of. For example, physical observation of a person's movement allows an attacker to learn the person's location, regardless whether the attacker is authorized to access this information.

2.8 Related Work

In this section, we discuss related work in the areas of people-location systems, information-representation schemes, access-control logics, and trust-management systems.

2.8.1 People-Location Systems

There are lots of people-location systems; Hightower and Borriello [64] give a comprehensive survey. Many of the systems have been developed to study and evaluate a particular

technology for locating people, not to investigate access-control issues that arise when different services for locating people, potentially run by different organizations, are brought together.

Let us discuss a few exceptions. In Myles et al.'s location system [96], a middleware service runs access control on behalf of individual location services. We discussed the drawbacks of a centralized solution in Section 2.3.1. In Spreitzer and Theimer's location system [117], each individual has a personal agent, which gathers location information about the individual and which implements access control to this information. While avoiding a centralized bottleneck, this solution has the drawback that it puts the load of gathering any statements required for the access decision on these agents. This task can be expensive, especially when access rights are forwarded. The system is designed to work in an environment with different administrative entities, although the actual implementation runs only within a single entity, and the authors do not mention how users specify services that they trust. Unlike our system, the location policy of a user is always specified by the user, and the system does not have the flexibility offered by digital certificates. Leonhardt and Magee's system [85] also suffers from this weakness. It relies on policy matrices, which are tedious to define.

2.8.2 Information-Representation Schemes

In this section, we investigate the information-representation schemes used in other pervasive computing projects and in other client-based access-control architectures.

Some pervasive computing projects [93, 5, 42] do not elaborate on the representation and naming schemes for the resources to which access is controlled. They typically assume that a resource is a physical resource, such as a printer, rather than information.

Pervasive computing projects that focus on controlling access to an individual's information [66, 39, 51] employ user-centric access policies. For each individual, there is an access policy that defines who can access the individual's personal information, such as her location information. This approach is similar to ours in that the information-representation scheme is made globally unique by distinguishing between the individual the information is about and the type of information. These projects do not elaborate on the establishment of access policies. Typically, the assumption is that an individual defines policies for information about her. In our representation scheme, by differentiating between the owner of information and the individual the information is about, we can formalize the (distributed) establishment of access policies (implemented as access rights) and easily support scenarios where an individual does not own her information.

Bauer et al. [16] and Howell and Kotz [69] present client-based access-control frameworks targeted at Web environments. Their information-representation scheme consists of a single string, containing the URL of a piece of information. A URL has the advantage that it can be globally unique. The disadvantage is that a URL is bound to the service providing the information described by the URL. If multiple services offered the same information about an individual, the owner of this information would have to establish access rights for each of them. Another disadvantage is that the representation scheme implicitly assumes that information is represented as (globally unique) URLs, but there is no explicit requirement that enforces this property. Therefore, it is possible to define access rights that grant access to non-unique information, which is problematic. In our representation scheme, we achieve global uniqueness by including the owner of information, not the service offering information. In addition, we make this owner an explicit part of the scheme and do not assume its implicit presence.

2.8.3 Access-Control Logics

Apart from Lampson et al.'s theory of authentication, there are many other logics for formalizing access control. Even when we restrict ourselves to logics targeted at pervasive computing, the list remains impressive. For example, there are REI [74], COBRA-ONT [36], SOUPA policy ontology [37], Cerberus' logic [5], environment roles [42], E-Wallet's OWL-based [45] rules [51], and Myles et al.'s P3P-based [44] rules [96]. However, most of them support only the encoding of an access policy, but not the encoding of credentials that show that an entity fulfills an access policy. Having only the encoding of an access policy can be sufficient in a centralized access-control architecture. However, a distributed access-control architecture, which is what we want for pervasive computing, requires credentials so that the access-control load can be shared and multiple entities can issue statements related to access control.

Apart from the speaks-for logic, there are a few other (application-specific) access control logics that support encoding of credentials [7, 16, 72, 93]. Some of them [7, 16] are similarly expressive as the speaks-for notation, which is a propositional modal logic. The logics allow the modeling of beliefs, which makes it straightforward to incorporate, for example, communication channels between entities or forwarding of access rights into the logic. Other logics [72, 93, 87] are based on (a subset of) first-order logic, which makes it possible to issue multiple access rights in a single statement. For instance, there needs to be only a single statement in order to grant all office owners access to the videostream from a camera in their offices. Propositional logic requires a statement for each office owner. SD3 [72] supports the encoding of credentials by including the public key of the

entity establishing a relation in the global name of the relation. RT [87] pursues a similar concept. In Minami and Kotz's logic [93], an entity keeps meta rules that define what other entities the entity trusts to evaluate its rules or to establish facts.

We choose the speaks-for logic as the foundation for our formal model since it is well known. However, we emphasize that all the concepts presented in this thesis could also be incorporated into an access-control logic based on first-order logic, if required.

2.8.4 Trust-Management Systems

Howell and Kotz's access-control framework [69] exploits SPKI/SDSI certificates [47] for expressing access rights. SPKI/SDSI certificates, together with mechanisms for the specification and evaluation of these certificates, are an example of a *trust-management system*. Trust management is a "unified approach to specifying and interpreting security policies, credentials, and relationships; it allows direct authorization of security-critical actions" [22]. We could have replaced SPKI/SDSI with an alternative trust-management system, such as PolicyMaker [21], KeyNote [22], Active Certificates [28], SD3 [72], RT [88], or X.509 Proxy Certificates [128]. (We list SD3 and RT both in this and the previous section because they come with a distinct formal model, which does not apply to the other systems.) KeyNote and X.509 Proxy Certificates are as powerful as SPKI/SDSI certificates. PolicyMaker and Active Certificates are more powerful since an access condition can be expressed in a general-purpose programming language. SD3 has built-in support for automatic retrieval of certificates from remote hosts. We decided to use SPKI/SDSI certificates since they are standardized and, slightly extended, are sufficiently powerful to implement our formal model. (SPKI/SDSI certificates have advantages when compared with alternatives, such as transitivity control and group-based access rights, which, however, are not relevant for this thesis.)

2.9 Summary

In this chapter, we introduced access policies, which define who should be able to access what confidential information under what constraints. With the help of access rights, we achieve the distributed specification of access policies. We also reviewed four architectures for the enforcement of access policies, in particular, centralized, service-based, client-based, and end-to-end access control.

We presented two specific sample implementations. The first one is employed for

securing access to a people-location system, the second one is a more general, client-based architecture targeted at pervasive computing. Its underlying formal model is based on Lampson et al.'s theory of authentication. We will use this architecture and the formal model as a base for addressing the remaining challenges to access control in the context of pervasive computing.

In the next chapter, we discuss how we capture the semantics of complex information for access control and how we reduce the number of access rights that individuals need to define.

Chapter 3

Information Relationships

In pervasive computing, there will be a multitude of services that provide potentially confidential information about an individual, such as her location, her personal files, her email, her calendar, or her activity. Some of this information might be offered by multiple services. In addition, a person might be a member of multiple environments over time. Therefore, having an individual issue access rights per client to be granted access, per service, per environment, and per type of information is not scalable. Role-based access control [100] addresses the first axis. Our information-representation scheme introduced in Section 2.5.1 allows for service-independent and environment-independent access rights. In this chapter, we concentrate on the fourth axis and examine ways to limit the number of types of information for which access rights need to be issued. To achieve this goal, we exploit *relationships* between information for access control. Information relationships are also useful for controlling access to complex information, such as a calendar entry that reveals location and activity information, since they can capture the semantics of this information [63].

3.1 Overview

Information relationships reduce the number of access rights that an individual needs to define. Consider the case of Alice managing access rights to her personal information, such as her location or her activity information. In a naïve solution, whenever she wants to grant someone access to all her personal information, she has to issue a separate access right for each type of personal information. In a better solution, Alice can bundle these different types of information and grant access rights to information bundles. When she

wants to grant someone access to her personal information, she now has to issue only a single access right. By bundling information, Alice establishes information relationships. The access-control mechanism exploits these relationships in order to derive individual access rights.

Another example demonstrating the usefulness of information relationships involves complex information. If the current entry in Carol's calendar says that she is having a meeting with Bob, only people who are at least allowed to access Carol's and Bob's location and activity information should have access to the calendar entry. To implement this rule, Carol should grant Alice an access right to this entry only if Alice already has access rights to Carol's and Bob's location and activity information. However, this approach is tedious and might lead to consistency problems if Bob revoked an access right to his location or activity information. Instead, access control should be aware of the semantics of information and know that calendar information contains location and activity information. We can use information relationships to capture the semantics. In the example, calendar information is related to location and activity information, and access control should be able take this relationship into account.

In this chapter, we make a client-based access-control architecture aware of information relationships that are common and important in pervasive computing. This way, we can run access control as often as possible in a fully distributed fashion. For more complex information relationships, it is possible to employ a centralized rule engine to reason about information. In particular, our contributions include

- the concept of information relationships as a first-class citizen in a client-based access-control architecture,
- a formal model for incorporating relationships into access control,
- a prototype implementation based on this model, and an evaluation of this implementation.

We present three types of relationships that are important for pervasive computing (Section 3.2) and incorporate these relationships into our formal model of distributed access control (Section 3.3). We discuss trade-offs between access rights and information relationships (Section 3.4). While individuals can define their own information relationships, we also give them the option to exploit information relationships defined by a standardization organization (Section 3.5). We examine how clients reason about relationships in proof building (Section 3.6), discuss a prototype implementation (Section 3.7), and analyze its complexity and performance (Section 3.8). Finally, we examine an alternative

implementation of information relationships in the context of Bauer's proof-carrying authorization [13] (Section 3.9).

3.2 Information Relationships

An information relationship states that a client should be granted access to a requested information item if the client already has access rights to information item(s) related to this item. Since a relationships affects access control to the requested information item, only the owner of this item can establish relationships for this item. We now describe a set of information relationships that are particularly relevant to pervasive computing.

Bundling-based relationships: Though there might be many different types of information about an individual, some of them have identical access requirements. An individual should be able to bundle such information and to issue only a single access right for the entire bundle. For example, assume that Alice wants to grant multiple people access to both her location and her activity information. Therefore, for each person, she needs to issue two access rights. Instead, Alice should be able to bundle her location and activity information in her personal information and to grant each person only a single access right to her personal information. Access control will then derive individual access rights for Alice's location and activity information from the bundle. In this way, we reduce the number of access rights that Alice needs to establish and the possibility of mistakes and information leaks.

Only Alice should be able to bundle her location information in other information. If we allowed Bob to bundle Alice's location information, he could bundle this information in his own personal information. Since Bob has access to his personal information, he would also be granted access to Alice's location information.

Combination-based relationships: Complex information is information that reveals other types of information. For example, a map shows the location of multiple people or Carol's calendar entry provides her location and the location of Bob, who is attending a meeting with Carol. Based on this relationship between the complex and the revealed pieces of information, it should be possible to make access rights to the complex information depend on access rights to the pieces. For example, a client can access a map only if the client has access rights to all the people's location shown on the map. Similarly, a client can access Carol's calendar only if the client has access rights to Carol's and Bob's location information.

Only Carol should be able to define a relationship for her calendar entry. In particular, it is up to Carol to decide whether she wants to define such a relationship in the first place. If Bob agrees to a meeting with Carol, he will have to rely on Carol not to make this information publicly available. If Carol is malicious, she will not respect Bob's privacy and let anyone access the corresponding calendar entry (or she will exploit other channels for providing the information in this entry). Only laws and regulations can avoid this information leak. However, if Carol is well behaved, she will want to respect Bob's privacy and she will want to take his access rights into account when deciding about whom to grant access to the calendar entry. Combination-based relationships make it easy to incorporate Bob's access rights since Carol does not even need to know about these access rights. We discuss tradeoffs between defining an access right to complex information and a relationship for the same information in more detail in Section 3.4.

Granularity-based relationships: Some information, such as location information, has different levels of granularity. There is an information relationship between the different levels: access rights to coarse-grained information should be derivable from access rights to fine-grained information. For example, if Bob had an access right to Alice's fine-grained location information (e.g., "Wean Hall 8220"), he automatically should also have an access right to Alice's coarse-grained location information (e.g., "Wean Hall"). There should be no need for Alice to establish the second access right.

With the exception of combination-based relationships, information relationships are static and require few updates by the individuals defining them. This property obviously holds for granularity-based relationships. For bundling-based relationships, we expect an individual to organize her information once and to make only a few changes after that. (If an individual did not want to define her own relationships, she could exploit global relationships, see Section 3.5.) Therefore, defining bundling-based information relationships will be a rare task.

We envision that services will automatically define combination-based relationships on behalf of individuals. For example, when Carol enters a calendar entry, the calendar application will automatically issue the corresponding relationship. Therefore, the potentially dynamic nature of combination-based relationships does not affect individuals.

Our solution has the benefit that it decreases the number of access rights that an individual needs to specify. This reduction becomes even more important when access rights are not established once and then left unchanged, but when they are defined in a dynamic way. For example, Alice might want to grant access to Bob only for a short period of time

in order to enable Bob to fulfill a task. If access rights are dynamic, information relationships will decrease the number of access rights that Alice needs to establish on an ongoing basis.

An obvious question is whether the three information relationships discussed in this section are common and important. We consider our approach of running access control by relating information as a dual to running access control by relating people (i.e., role-based access control [100]). For each type of information relationship, there exists a corresponding, major concept in role-based access control. In particular, bundling-based relationships correspond to assigning roles to people. Combination-based relationships are similar to resources that require the presence of multiple roles. Granularity-based relationships are similar to hierarchical role schemes. This observation suggests that the three relationships are common and sufficient, and so far this has held up in our examples.

There are other concepts in role-based access control, such as separation of duty, which ensures that a person cannot be a member of conflicting roles, or sessions, which regulate role activation. While it is possible to come up with information relationships that resemble these concepts, we strive to keep information relationships simple. This goal is based on the observation that, in pervasive computing, access rights and information relationships will often be managed by individuals. Role-based access control assumes the existence of dedicated administrators with a thorough understanding of access control. This assumption will probably not hold for pervasive computing.

3.3 Formal Model

In this section, we extend the formal model introduced in Section 2.5.2 to support the information relationships introduced in the previous section. We conclude by demonstrating the application of the model in a more involved scenario.

An information relationship implies that if a principal, B, has access to some information items, $E_i.x_i$, B should also have access to a (related) item, $D.x$. Formally, we use $E_1.x_1 \otimes E_2.x_2 \otimes \ldots \longrightarrow D.x$ for expressing this relationship, and we want the axiom

$$\begin{aligned} &\vdash (E_1.x_1 \otimes E_2.x_2 \otimes \ldots \longrightarrow D.x \\ &\wedge B \stackrel{E_1.x_1}{\Longrightarrow} A_1 \ \wedge B \stackrel{E_2.x_2}{\Longrightarrow} A_2 \wedge \ldots) \\ &\supset (B \stackrel{D.x}{\Longrightarrow} C) \end{aligned} \tag{3.1}$$

to hold for certain conditions on the principals E_i, their information $E_i.x_i$, and the princi-

pals A_i and C. In Sections 3.3.1, 3.3.2, and 3.3.3, we show that instantiating Axiom (3.1) in different ways straightforwardly leads to the concepts of bundling-based, combination-based, and granularity-based information relationships, respectively. For each type of relationship, we also discuss plausible conditions on the various entities in the axiom and pick the most useful one.

In addition to formalizing the application of information relationships in access control, we also need to formalize their establishment. Since access to the information items on the left-hand side of a relationship also grants access to the item on the right-hand side, only the principal owning the information on the right-hand side or a principal speaking for that principal regarding the information should be able to establish the relationship, or

$$\begin{aligned} &\vdash (F \textbf{ says } (E_1.x_1 \otimes E_2.x_2 \otimes \ldots \longrightarrow D.x)) \wedge (F \overset{D.x}{\Longrightarrow} D) \\ &\supset (E_1.x_1 \otimes E_2.x_2 \otimes \ldots \longrightarrow D.x). \end{aligned} \tag{3.2}$$

For example, only Alice as the owner of her location information can bundle this information in her personal (or some other) information. Similarly, only Carol as the owner of her calendar can define a relationship stating that anyone who has access to her (and potentially other people's) location and activity information can also access her calendar entry.

We now look at useful instances of Axiom (3.1).

3.3.1 Bundling-Based Relationships

Limiting the number of information items on the left-hand side of Axiom (3.1) to one item corresponds to the concept of bundling-based relationships. For example, the statement Alice.personal $\longrightarrow$ Alice.location denotes that information Alice.location is bundled in information Alice.personal, that is, if someone has access to Alice's personal information, he should also have access to her location information.

There are two plausible conditions on the various entities in Axiom (3.1). The first one requires $A_1 = C$, or

$$\vdash (E_1.x_1 \longrightarrow D.x \wedge B \overset{E_1.x_1}{\Longrightarrow} A_1) \supset (B \overset{D.x}{\Longrightarrow} A_1). \tag{3.3}$$

The second one is $A_1 = E_1$ and $D = C$, or

$$\vdash (E_1.x_1 \longrightarrow D.x \wedge B \stackrel{E_1.x_1}{\Longrightarrow} E_1) \supset (B \stackrel{D.x}{\Longrightarrow} D).$$

We choose the first condition since the second one is too limiting. For example, given $E_1.x_1 \longrightarrow D.x$, $B \stackrel{E_1.x_1}{\Longrightarrow} C$, and $C \stackrel{D.x}{\Longrightarrow} D$, it should be possible to conclude $B \stackrel{D.x}{\Longrightarrow} D$, which the second condition does not permit.

Our formal model allows individuals to bundle some of their information in someone else's information. For example, people collaborating on a project can bundle information that is relevant to the project in information owned by the project manager. The project manager can then grant access to the information bundle.

Our discussion assumes that an individual establishes all her bundling-based relationships. However, many individuals might establish the same types of relationships. Therefore, it should be possible for a standardization organization to establish global bundling-based relationships, to which individuals can subscribe. We discuss this concept in Section 3.5.

3.3.2 Combination-Based Relationships

Not limiting the number of information items on the left-hand side of Axiom (3.1) corresponds to the concept of combination-based relationships. For example, the statement Alice.location $\otimes$ Bob.location $\longrightarrow$ Map_Service.map denotes that the map offered by a map service can be accessed by anyone who has access to both Alice's and Bob's location information. (The assumption is that only Alice's and Bob's location is shown on the map.)

Again, there are two plausible conditions on the various entities in Axiom (3.1). We can require $A_1 = E_1$, $A_2 = E_2$,..., and $C = D$, or

$$\begin{aligned} &\vdash (E_1.x_1 \otimes E_2.x_2 \otimes \ldots \longrightarrow D.x && (3.4)\\ &\wedge B \stackrel{E_1.x_1}{\Longrightarrow} E_1 \ \wedge B \stackrel{E_2.x_2}{\Longrightarrow} E_2 \wedge \ldots)\\ &\supset (B \stackrel{D.x}{\Longrightarrow} D). && (3.5)\end{aligned}$$

The second option is $A_1 = A_2 = \ldots = C$, which is an extension of the condition for bundling-based relationships, or

$$\vdash (E_1.x_1 \otimes E_2.x_2 \otimes \ldots \longrightarrow D.x$$
$$\wedge\, B \overset{E_1.x_1}{\Longrightarrow} C \;\wedge B \overset{E_2.x_2}{\Longrightarrow} C \wedge \ldots)$$
$$\supset (B \overset{D.x}{\Longrightarrow} C).$$

This condition requires a single entity that has access to all of $E_1.x_1$, $E_2.x_2$,..., and $D.x$, through which B would then acquire its access rights. However, this requirement is unrealistic in practice and does not fit the intuitive model of combination-based relationships. Therefore, we pick the first condition.

3.3.3 Granularity-Based Relationships

Granularity-based information relationships are a special case of bundling-based relationships. For example, Alice could define two types of location information, such as Alice.location_fine and Alice.location_coarse, and establish

$$\text{Alice.location\_fine} \longrightarrow \text{Alice.location\_coarse}.$$

However, requiring individuals to introduce separate types of information for different granularities is tedious and not intuitive. Instead, we observe that granularity-based access rights to information can be represented as an access right that has a constraint on the returned information. For example, if Bob had access to Alice's coarse-grained location information and asked for her location, the result would have to be coarse grained. This result (e.g., (Building Manager, Wean Hall 8220) using our information representation scheme) is itself an item about which there is some information (such as its granularity). Constraints are not a new concept in access control. For instance, an access right can be constrained to be valid only during office hours. By extending our formal model to support constraints, we can express that Bob has access to Alice's location at fine or coarse granularity as follows:

$$\text{Bob} \overset{\text{Alice.location}}{\underset{\text{?result.granularity} \geq \text{fine}}{\Longrightarrow}} \text{Alice}.$$

(For readability reasons, we put constraints under the arrow, the statement remains a speaks-for operator.)

Table 3.1: Example statements. Alice and Bob grant Carol and Dave access to their personal information using information relationships.

(1)	Alice **says** Alice.personal $\longrightarrow$ Alice.location [?result.granularity $\geq$ fine]
(2)	Alice **says** Carol $\overset{\text{Alice.personal}}{\Longrightarrow}$ Alice
(3)	Alice **says** Dave $\overset{\text{Alice.location}}{\underset{\text{?result.granularity=coarse}}{\Longrightarrow}}$ Alice
(4)	Bob **says** Carol $\overset{\text{Bob.location}}{\underset{\text{?result.granularity}\geq\text{fine}}{\Longrightarrow}}$ Bob
(5)	Location Service **says** Alice.location [?result.granularity = fine] $\otimes$ Bob.location [?result.granularity = fine] $\longrightarrow$ (Location Service, Wean Hall 8220).people

3.3.4 Example

In this section, we demonstrate the application of our formalism in an example scenario. The scenario involves a location service that provides the identity of the people in a room, two users Alice and Bob managing their access rights, and two users Dave and Carol trying to access Alice's or Bob's confidential information.

Alice bundles fine-grained location information (and potentially other) information in her personal information (Statement (1) in Table 3.1). She grants Carol access to her personal information (2) and Dave access to her coarse-grained location information (3). Bob grants Carol access to his fine-grained location information (4). The information provided by the location service should be accessible only if a client has access rights to the location information of all individuals in a room, that is, we require a combination-based relationship. Assuming that only Alice and Bob are in Wean Hall 8220, the service defines a corresponding relationship (5).

Carol queries for the people in Wean Hall 8220. She is granted access based on the information relationship of the location service (5), Alice's information relationship (1), Alice's access right (2), and Bob's access right (4). Dave also issues a query and is denied access, since the intersection of ?result.granularity = coarse (3) and ?result.granularity = fine (5) is empty.

As demonstrated in this section, our scheme makes it straightforward to run access control based on information relationships. Moreover, there is no need for the different types of information to be owned by the same entity.

3.4 Discussion

There are two options for granting a principal access to complex information. First, the owner of the information can give the principal an access right to the information, which lets the principal access the information in an unrestricted way. Second, the owner can define a combination-based relationship for the information, which allows the principal to access the information if the principal can access all the other information revealed by the complex information. Note that the owner can use both options simultaneously. In this section, we discuss trade-offs between the two options. Furthermore, we investigate an alternative approach, in particular, selective disclosure of complex information

Let us illustrate the trade-offs with an example. Carol grants her executive assistant, Alice, an access right to her calendar, or

$$\text{Carol}\ \mathbf{says}\ \text{Alice} \xRightarrow{\text{Carol.calendar}} \text{Carol}.$$

Furthermore, Carol defines a combination-based relationship that lets principals access her calendar entry if they can access the location and activity information revealed by the entry, or, assuming that Carol is currently meeting with Bob,

$$\text{Carol}\ \mathbf{says}$$
$$(\text{Carol.location} \otimes \text{Carol.activity} \otimes \text{Bob.location} \otimes \text{Bob.activity} \longrightarrow \text{Carol.calendar}).$$

In this scenario, Alice has unrestricted access to Carol's calendar entry, whereas other principals can access this entry only if they have access to Carol and Bob's location and activity information. The information relationship prevents clients other than Alice from learning information revealed by the complex information in case they do not have access rights to the revealed information. However, since Alice has an explicit access right to Carol's calendar entry, Alice could learn Bob's location or activity information without being in the possession of an access right granted by Bob to this information. We could avoid this information flow by forbidding owners of complex information to issue access rights to complex information. Nonetheless, we support this option because an owner of information could circumvent the restriction by using distribution channels our system is not aware of. Furthermore, our example reflects already existing, widely accepted social behavior. In particular, executive assistants are typically expected to have access to their boss' calendar, and people meeting with their boss know that the executive assistant can

access the calendar. In pervasive computing, we use this social behavior as a guide for designing our security mechanisms. This approach is consistent with the second privacy design guideline issued by the European Disappearing Computer Initiative [79]. In particular, the guideline says:

> *Search for existing solutions in the physical world or in old systems for the similar class of problem/service, and understand the way in which new technologies change the effects of classic issues.*

Finally, if we tried to prevent owners of complex information from issuing access rights to the information, the usefulness of a service could be drastically reduced, and individuals could be tempted to use other services that lack access control entirely instead. We emphasize that these observations apply specifically to pervasive computing, but not necessarily to other environments (e.g., military or financial services).

When agreeing to meet with Carol, Bob needs to trust Carol to properly deal with his location or activity information during the time of the meeting. For example, Bob trusts Carol not to grant anyone an access right to her calendar entry. To establish this trust, Carol could make Bob aware of access rights that she issued to her calendar entry. If Bob did not agree with Carol's choices, he could ask Carol not to list his identity in her calendar entry or he could refrain from meeting with her. Furthermore, for auditing reasons, the calendar service could notify Bob whenever there is a query for the calendar entry in question.

Access control based on a combination-based relationship requires that a client has access to all types of information listed on the left-hand side of the relationship. As an alternative, a service providing complex information could mask information to which a client does not have access rights and selectively grant access to parts of the complex information. However, this approach can cause information leaks based on statistical inference. Assume that Bob is meeting with Carol in Wean Hall 8220. Dave has access to Carol's location and activity information. He can also access Bob's location information. Dave then queries Carol's calendar entry and submits the access rights issued by Carol. The calendar service masks Bob's identity, and Dave learns that Carol is in Wean Hall 8220 and that Carol is in a meeting. Next, Dave queries Carol's entry a second time and submits the access right issued by Bob. The calendar service masks Carol's identity and Bob's activity, and Dave learns that Bob is in Wean Hall 8220. Dave can now combine the results of the two queries and infer that Bob is in a meeting. Since Dave does not have an access right to Bob's activity information, there is an information leak.

By not masking information, we can avoid this information leak for a single service. However, Dave might be able to infer the same knowledge by using multiple location and

activity services, as illustrated in Section 2.7. This leak is difficult to avoid for multiple services; we do not address it in this thesis.

3.5 Global Information Relationships

Many individuals might define the same kind of bundling-based relationships. We present two approaches that allow an individual to exploit global bundling-based relationships, instead of having to define her own relationships. For both approaches, we assume that a global relationship is issued by a standardization organization on behalf of an individual. There can be many such organizations. The approaches differ in the requirements that need to be satisfied in order for a global relationship to become valid.

The first approach has the advantage that it does not require any changes to the existing formal model, but it gives a lot of power to a standardization organization. The second approach does not suffer from this drawback, but it requires an extension to the formal model.

3.5.1 Delegation-Based Approach

In the first approach, in order for a global relationship to become valid, an individual needs to grant the standardization organization access to her information. We call this approach "delegation-based" since the individual delegates all her capabilities to decide about the information's access rights and relationships to the organization. We illustrate the approach with an example. Alice lets organization ACME decide the composition of her personal information. ACME decides that medical information is part of personal information and issues

$$\text{ACME } \mathbf{says} \ (\text{Alice.personal} \longrightarrow \text{Alice.medical}). \tag{3.6}$$

In addition, Alice grants the organization access to her medical information, or

$$\text{Alice } \mathbf{says} \ (\text{ACME} \overset{\text{Alice.medical}}{\Longrightarrow} \text{Alice}). \tag{3.7}$$

This statement defines a speaks-for relationship between ACME and Alice regarding Alice.medical, which we can exploit to establish Alice.personal $\longrightarrow$ Alice.medical from Statement (3.6) according to Axiom (3.2).

ACME bundles Alice.medical in Alice.personal. This approach works well under the assumption that both ACME and Alice agree that the composition of Alice.personal is managed by ACME. However, it could break if there was no such agreement. For example, it is possible that ACME bundles Alice.medical not only in Alice.personal, but also in other information owned by Alice (e.g., in Alice.health). Therefore, an access right to Alice.health would also grant access to Alice.medical. Alice would not want this property if she was not aware of this additional relationship and assumed that only she managed the composition of Alice.health.

To avoid this conflict, ACME should not bundle information owned by Alice in other information owned by Alice. Instead, ACME should bundle the information in information owned by ACME. Formally,

$$\text{ACME } \mathbf{says} \ (\text{(ACME, Alice).personal} \longrightarrow \text{Alice.medical}). \tag{3.8}$$

To allow Alice to issue access rights to her personal information, ACME needs to state

$$\text{ACME } \mathbf{says} \ (\text{Alice} \xRightarrow{\text{(ACME, Alice).personal}} \text{ACME}). \tag{3.9}$$

Alice can exploit ACME's definition of her personal information by issuing an access right to (ACME, Alice).personal. If she did not want to use ACME's definition, she could issue an access right to her own personal information, that is, Alice.personal.

The approach requires that Alice grants ACME access to each type of information that is part of ACME's notion of personal information (e.g., Statement (3.7)). However, Alice might not know for what information she has to issue such a statement, because she does not know or does not want to know what information is part of ACME's notion of personal information. (The whole point of letting ACME decide about relationships is to remove these decisions from Alice.) Similarly, Statements (3.8) and (3.9) assume that ACME issues an information relationship and an access right for each individual that wants to use ACME's relationships. However, if ACME was a standardization organization, it might be oblivious to the individuals who are using its specifications. The two observations suggest that digital certificates implementing these three statements should allow for wildcards. For example, in a certificate expressing Statement (3.7), Alice should be able to list a wildcard (or a set of possible values) instead of "medical". Similarly, in certificates expressing Statements (3.8) and (3.9), ACME should be able to include a wildcard, instead of "Alice".

3.5.2 Naming-Based Approach

The first approach hurts the principle of least privilege [108], since it grants access rights to the standardization organization, but the organization does not actually need to access this information. Our second approach does not have this disadvantage. It gives the organization the capability to define a bundling-based relationship for information owned by an individual, but it does not grant access rights to the organization. Therefore, we call this approach "naming-based". Its drawback is that it requires the introduction of an additional axiom to the formal model.

In terms of the example introduced in Section 3.5.1, we use

$$\text{ACME} \overset{\{\text{“bundle Alice.medical”}\}}{\Longrightarrow} \text{Alice}$$

to denote that ACME can establish relationships for Alice.medical. Only Alice and principals speaking for Alice regarding {"bundle Alice.medical"} can establish this statement. Remember that $B \overset{A.x}{\Longrightarrow} A$ is just an abbreviation for $B \overset{\{\text{“read } A.x\text{”}\}}{\Longrightarrow} A$.

Formally,

$$\begin{aligned} &\vdash \text{ACME} \overset{\{\text{“bundle Alice.medical”}\}}{\Longrightarrow} \text{Alice} \\ &\wedge\ \text{ACME}\ \mathbf{says}\ (\text{Alice.personal} \longrightarrow \text{Alice.medical}) \\ &\supset (\text{Alice.personal} \longrightarrow \text{Alice.medical}). \end{aligned}$$

In this way, ACME can define a bundling-based relationship for Alice.medical, without having an access right to this information. Note that the owners of the two types of information in the relationship must correspond to Alice. In particular, ACME must not be allowed to bundle Alice.medical in its own information. This relationship would grant ACME access to Alice.medical, which we want to avoid.

Allowing ACME to bundle Alice.medical in information that is owned by Alice can result in the problem discussed in Section 3.5.1, that is, ACME bundles information in some of Alice's information, without Alice being aware of. There, we worked around the problem by having ACME bundle information owned by Alice in information owned by ACME. As mentioned above, this is not an option here. Instead, Alice and ACME can adopt a naming scheme for information that avoids disagreement between Alice and ACME about who is responsible for the composition of some of Alice's information.

For example, ACME could prefix the type of information with its name, that is, instead of bundling Alice.medical in Alice.personal, ACME could bundle Alice.medical in Alice.ACME_personal.

Similar to the first approach, in a digital certificate expressing the permission to define relationships, a wildcard could be used instead of "medical". Furthermore, it is possible to limit the scope of a certificate. For example, it could allow bundling of information in Alice.ACME_personal, but not in other information owned by Alice.

3.6 Proof Building

In our distributed access-control architecture, we have a client ship a proof of access to a service. As mentioned in Section 2.6.2, a proof of access needs to show that a client speaks for the owner of the requested information regarding this information. In that section, proofs of access consisted solely of combined access rights (and instructions on their combination). However, when access control exploits information relationships, proofs of access can also contain relationships. Figure 3.1 illustrates a proof of access with an information relationship. The structure of a proof corresponds to the types of axioms required for validating the proof.

As shown in Section 3.3, for each type of information relationship, Axiom (3.1) is instantiated in a different way. Another characteristic that differentiates the relationships from each other is how they are used in proof building. We now discuss proof building for each relationship.

3.6.1 Bundling-Based Relationships

We first illustrate how proof building exploits bundling-based relationships. A client collects access rights and information relationships received from individuals. It stores speaks-for statements derived from access rights or from other speaks-for statements in a graph where nodes represent principals and edges represent information. For a request, the algorithm traverses the graph in a breadth-first way to find a proof of access, starting with the owner of the requested information and ending at the client. During this graph traversal, when the information in a candidate edge does not match the requested information, the algorithm looks at all bundling-based relationships that have the requested information on their right-hand side and checks whether the left-hand side of the relationship matches the information in the candidate edge. (If there is a hierarchy of bundling-based relation-

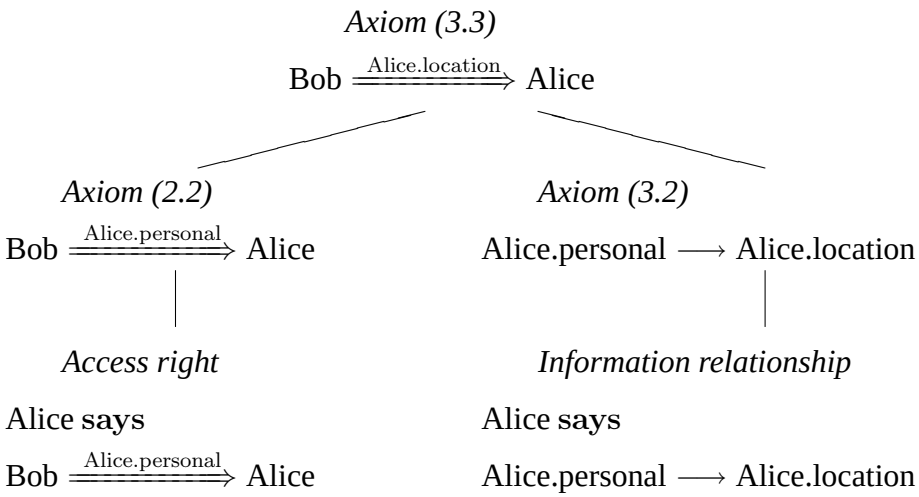


Figure 3.1: Structured proof. The proof shows that Bob can speak for Alice regarding her location information, based on an access right to Alice's personal information and an information relationship between Alice's personal and her location information.

ships, this step requires exploration of multiple relationships.) If so, the algorithm accepts the candidate edge and the edges connected to this edge become new candidate edges. Figure 3.2 illustrates this algorithm.

The algorithm looks at each edge at most once. Therefore, if there are n speaks-for statements and no information relationships, its worst-case complexity is $\mathcal{O}(n)$. If there are also m information relationships, the algorithm will look at an information relationship at most once for each candidate edge. The worst-case complexity becomes $\mathcal{O}(nm)$. Although complexity is multiplicative, we expect proof building to be practical. First, the absolute value of n will be larger for the case without information relationships, since this case requires separate access rights for information with identical access requirements. Second, we expect principals to define information relationships in a way such that information is bundled only in a small number of information bundles and to keep the hierarchies of bundling low. We measure proof-building time in Section 3.8.2.

Global relationships could make proof building more complex since they require additional speaks-for statements for their establishment. For example, establishing the relationship in Statement (3.8) requires a statement that shows that ACME can speak for Alice regarding Alice.medical. If a client has to locate the corresponding access right in its collection of access rights and information relationships during proof building, proof building

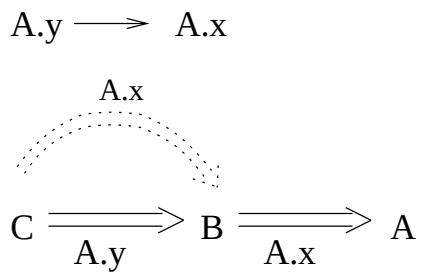


Figure 3.2: Proof building. The goal is to prove "$C \stackrel{A.x}{\Longrightarrow} A$". The search starts at A. It locates "$B \stackrel{A.x}{\Longrightarrow} A$" and tries to prove "$C \stackrel{A.x}{\Longrightarrow} B$". The search locates "$C \stackrel{A.y}{\Longrightarrow} B$" and uses "$A.y \longrightarrow A.x$" to get the required information.

will become more complex. However, it is possible to prevent this increase in complexity. Namely, a client should insert a statement defining a global relationship into its collection of access rights and relationships only if the relationship can actually be established. In this way, we ensure that there are only valid information relationships in the collection. Therefore, the client should locate the access right that establishes the relationship when it inserts the global relationship into its collection, not during proof building.

3.6.2 Combination-Based Relationships

We do not expect clients to exploit combination-based relationships in proof building. Instead, these relationships are used by services. A combination-based relationship is tightly coupled to the information that a service provides (e.g., for a map service, the relationship consists of all the people shown on the map). Therefore, it is straightforward for a service to establish the corresponding relationship and to exploit it for access control. In particular, for each information item on the left-hand side of the relationship, the service has the client build a proof of access. The service then aggregates these individual proofs of access and its combination-based relationship into a summary proof.

Proof building by a client can be difficult in this scenario since the client might not know how the individual proofs of access should look like and the service cannot inform the client of their nature without leaking information. For example, if a map service had the information relationship Alice.location $\otimes$ Bob.location $\longrightarrow$ Map_Service.map, a client would have to build proofs of access for Alice.location and Bob.location. However, the client might not know who is on the map, and the service cannot tell it without leaking location information about Alice and Bob. We present a solution for this problem in Chap-

ter 6.

3.6.3 Granularity-Based Relationships

For granularity-aware access control, the algorithm introduced in Section 3.6.1 must be extended to take constraints on access rights and information relationships into account. For example,

$$\text{Carol} \overset{\substack{\text{Alice.location} \\ \text{?result.granularity} \geq \text{fine}}}{\Longrightarrow} \text{Bob}$$

and

$$\text{Bob} \overset{\substack{\text{Alice.location} \\ \text{?result.granularity} = \text{coarse}}}{\Longrightarrow} \text{Alice}$$

can be concatenated to

$$\text{Carol} \overset{\substack{\text{Alice.location} \\ \text{?result.granularity} = \text{coarse}}}{\Longrightarrow} \text{Alice}.$$

Since granularity-based relationships can be directly encoded in access rights instead of requiring separate information relationships, there will be no extra information relationships, which reduces the complexity of proof building.

In summary, our proof-building algorithm is simple, but it has proved sufficient for our application scenarios. We can also trade off computation vs. storage. Instead of computing a proof upon a request, a client can store the closure of its access rights and update this closure whenever it receives an access right or information relationship. Alternatively, instead of employing a brute-force prover, we can use more sophisticated theorem provers. For example, Bauer et al. [16] employ the Twelf logical framework [104] for building proofs of access.

3.7 Implementation

Our implementation is based on the client-based access-control framework introduced in Section 2.6, to which we added support for building proofs of access involving information relationships, as outlined in Section 3.6.

```
(bundling-relationship
   (version ''1'')
   (issuer
      (information (pub_key:alice) alice location))
   (subject
      (information (pub_key:alice) alice personal)))
```

Figure 3.3: Bundling-based information relationship. Alice's location information is bundled in her personal information. (The digital signature is omitted.)

We use extended SPKI/SDSI certificates for expressing and storing bundling-based information relationships. We give an example in Figure 3.3. In the certificate, Alice's location information is bundled in her personal information. The certificate needs to be signed by the owner of the information in the **issuer** entry, that is, Alice. For some scenarios, such as global relationships, an entity other than this owner may sign the certificate. We currently do not support this feature. In general, SPKI/SDSI certificates do not support speaks-for statements where the principal making the statement does not correspond to the principal on the right-hand side of the speaks-for relationship (e.g., $A \, \mathbf{says} \, (B \stackrel{C.x}{\Longrightarrow} D)$.

3.8 Performance Analysis

We have deployed access control with information relationships in the context of Aura services that provide calendar and location information. In this section, we evaluate our implementation, both with measurements and analytically.

We analyze our proposed information relationships along three axes: their effect on the number of issued access rights, on proof-building time, and on request-processing time. We take our measurements on an unloaded Pentium IV/2.5 GHz with 1.5 GB of memory, Linux 2.4.20, and Java 1.4.2.

3.8.1 Number of Access Rights

We examine how bundling-based relationships affect the number of access rights that an individual has to issue. In particular, we compare the number of statements to be made in a world with information relationships to the corresponding number in a world without

information relationships.

Let us first consider the case with information relationships. Assume that there is a full, tree-based hierarchy consisting of l levels of bundling-based relationships, where $l = 1$ corresponds to the base case consisting of a root node and some leaf nodes. Each parent node has m children (i.e., m types of information are bundled in each parent node). There are k clients. Each of them is assigned to a single node in the hierarchy, meaning that the client is given an access right to the information covered by the node. There are different ways to distribute the clients in the hierarchy. In this scenario, regardless of the clients' distribution, the number of access rights to be issued is always k and the number of relationships is always $\sum_{i=1}^{l} m^i$. The first curve in Figure 3.4 shows the overall number of access rights and relationships for different values of l. We choose $k = 50$ and $m = 3$.

We want to know the number of access rights that would have to be issued if there were no relationships, but the k clients should have access to the same information as in the case with relationships. This number depends on the distribution of these clients in the tree. We examine three different distributions. The first one is artificial and presents the best case in possible savings of issued access rights: all k clients are assigned to the root node. Without relationships, this scenario would require km^l access rights (since there are m^l leaf nodes), as shown by the second curve in Figure 3.4. The second distribution is also artificial and presents the worst case in possible savings: all k clients are distributed randomly among the m^l leaf nodes. This scenario would require k access rights, as shown by the third curve. The third distribution is a more realistic one: we distribute the k clients evenly among the $l + 1$ layers of nodes in the hierarchy. This would require $\frac{k}{l+1} \sum_{i=0}^{l} m^i$ access rights, as shown by the fourth curve. As we can see in Figure 3.4, with the exception of the worst case, where relationships become unnecessary, relationships can lead to a significant decrease in the number of issued access rights and information relationships.

We believe that it is intuitive for individuals to define relationships since the underlying paradigm is well known and already used for organizing files in a filesystem or digital pictures in a photo album. If the individual did not want to define her own relationships, she could always exploit relationships defined by third entities.

Similar to bundling-based relationships, granularity-based relationships require individuals to issue fewer access rights, where the decrease is proportional to the available levels of granularities. Combination-based relationships also reduce the number of access rights since they prevent a service from having to issue separate access rights for complex information.

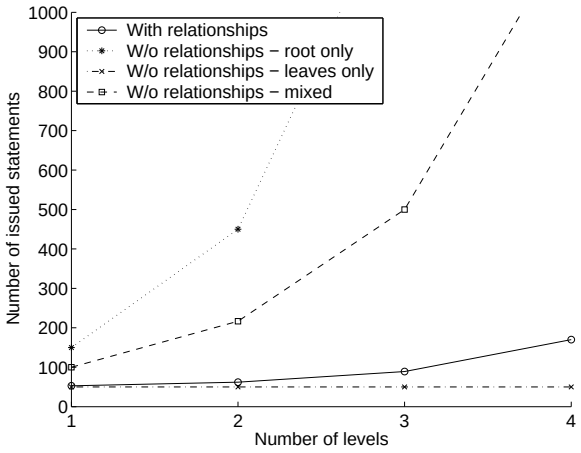


Figure 3.4: Number of issued statements. We compare the number of issued access rights and relationships in a world with relationships to the number of issued access rights in a world without relationships, for different levels of bundling-based relationships and distributions of clients in this hierarchy.

3.8.2 Proof-Building Time

We examine the cost of proof building, as explained in Section 3.6. Our experiment consists of locating a set of speaks-for statements in a pool of statements and then assembling them in a proof of access. We consider up to five sequentially connected, bundling-based relationships. Because of manageability reasons, we do not expect individuals to create more than five levels of bundling-based relationships in their information hierarchies.

Our experimental setup covers a worst-case scenario: We pick five among 50 possible principals and create a sequential path of access rights between them (i.e., a principal delegates its access right to the next principal in the path). All experiments will locate this path, since there is considerable variation for different paths. The results that we present are consistent with results for other paths. We then put the access rights into a pool, which we expand by adding random access rights. The random access rights form a directed graph where each recipient of an access right is (indirectly) reachable from the issuer of the first access right in the predetermined path and where none of the recipients is on this path (i.e., we do not allow shortcuts). We randomly set the information in each access right to one of the possible information items covered by the bundling-based relationships. An experiment is characterized by a number of random access rights and a number of relationships and is run ten times. Figure 3.5 reports the mean proof-

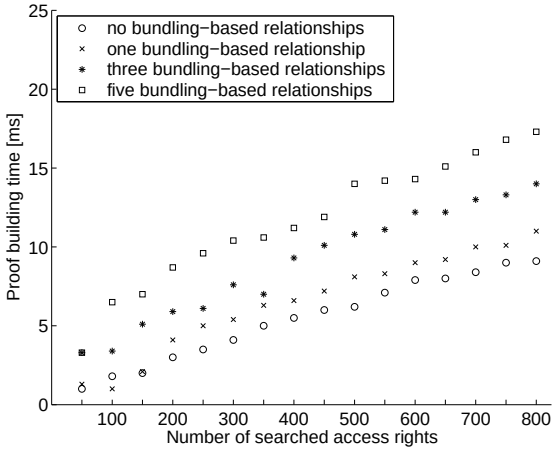


Figure 3.5: Proof-building time. The graph shows the mean proof-building time for different numbers of bundling-based relationships. For a fixed number, there is a linear increase in the number of searched access rights.

building time. According to Section 3.6, the worst-case complexity is multiplicative in the number of speaks-for statements and information relationships. Our results confirm that the increase is linear for a particular number of relationships.

In practice, information relationships do not necessarily lead to increased proof-building cost. In particular, if there are relationships, there typically will be fewer access rights in a client's pool, which makes proof building cheaper. Therefore, for a given value on the x-axis in Figure 3.5, the various proof-building times are not directly comparable. The actual cost strongly depends on the number and types of information relationships. Overall, as we will show in Section 3.8.3, the cost of building a proof is comparable to the cost of processing a request. We also observe that the numbers presented in Figure 3.5 cover a worst-case scenario. First, we require four access rights for the proof of access; we expect this number to be smaller in practice. Second, our experimental setup ensures that all access rights in a user's collection might be explored for proof building. In real collections, many of the access rights in a collection will not be explored. For example, a search for access rights to Alice's location information typically will not explore access rights to Bob's location information.

Table 3.2: Client-response time. Mean and standard deviation of elapsed time for security operations (in bold) and for gathering information [ms].

Entity	Step	μ	(σ)
Both	**SSL Socket creation**	53	(6)
Service	**Access control**	3	(2)
Service	Gather location information	56	(7)
	Total	129	(13)

3.8.3 Client-Response Time

We study the influence of bundling-based relationships on client-response time in our prototype implementation. As mentioned in Section 3.6, the other types of relationships do not have a negative influence on complexity. We measure the response time experienced by a client when retrieving location information from a service. We assume that the client contacts a service providing people-location information. This service fingers the Linux desktop computer of the queried individual and tries to determine the individual's location from her activity.

We run the client and the location service service on an unloaded host. In addition, there is another host that runs a database storing digital certificates and static information about users (e.g., their userids). The client and the service contact this database to retrieve any data required for generating a request for information or processing such a request.

Our asymmetric cryptographic operations required for setting up SSL connections and validating the signatures of certificates employ 1024 bit RSA keys. An experiment is run 100 times. We report the mean and standard deviation (in parentheses). Since we are not interested in proof-building time for these experiments, we assume that clients have pre-built proofs.

In the first experiment, Alice grants Bob access to her location information in an access right. Bob submits the corresponding proof to a location service, which validates it and locates Alice. The mean response time is 129 ms (13 ms). More detailed results are in Table 3.2. Access control takes only a few milliseconds, the main cost is validating the signature of the certificate. Gathering the location information is the most expensive step. Setting up an SSL connection requires two costly RSA decryption/signing operations for client and server authentication. Access control and setting up SSL are CPU bound and will benefit from faster hardware.

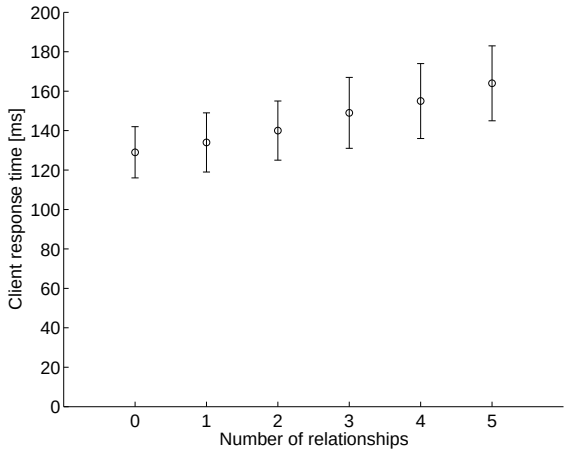


Figure 3.6: Client-response time. The elapsed time increases linearly with the number of bundling-based relationships.

In the second experiment, Alice grants Bob access to her location information via a variable number of sequentially connected, bundling-based relationships. The results are in Figure 3.6. Note that the scaling of the y-axis is different from the scaling in Figure 3.5. The figure shows that client-response time increases linearly by about 7 ms for each additional relationship. The main reasons for the increase are increased transmission cost and the validation of an additional signature.

3.9 Proof-Carrying Authorization

We have incorporated information relationships into Lampson et al.'s theory of authentication. It is possible to add information relationships to other access-control logics. As a proof of concept, let us explore this addition for Bauer's proof-carrying authorization framework [13]. This framework consists of an application-specific access-control logic, a higher-order logic, and a client-based access-control architecture targeted at Web environments. To prove soundness of the application-specific logic, Bauer gives the logic a semantics by defining all its operators in the higher-order logic, which is known to be sound. (A logic is sound if any provable statement is true in all semantic models of the logic.) Let us now take a closer look at a relevant subset of the application-specific logic and then incorporate information relationships into the application-specific logic.

3.9.1 Overview

The application-specific logic is similar to Lampson et al.'s theory of authentication. Principals can express beliefs with the **says** operator. The **goal** constructor describes the URL that a client wants to access. It has two parameters: a URL and a nonce to avoid replay attacks. A client, B, issues a request for URL by stating B **says** $(\mathbf{goal}(URL, nonce))$, where $nonce$ is given to the client by the contacted service. In order to be granted access, the client needs to prove that the owner of the information to which the URL is pointing, A, also issues this statement, or A **says** $(\mathbf{goal}(URL, nonce))$. The client can exploit two ways of delegation for this purpose: The **speaksfor** operator transfers all of a principal's access right to another principal, or

$$\begin{aligned}&\vdash (A \,\mathbf{says}\, (B \,\mathbf{speaksfor}\, A)) \wedge (B \,\mathbf{says}\, (\mathbf{goal}(URL, nonce)))\\ &\supset (A \,\mathbf{says}\, (\mathbf{goal}(URL, nonce))).\end{aligned} \tag{3.10}$$

The **delegate** operator transfers only the access right for a particular URL, or

$$\begin{aligned}&\vdash (A \,\mathbf{says}\, (\mathbf{delegate}(A, B, URL))) \wedge (B \,\mathbf{says}\, (\mathbf{goal}(URL, nonce)))\\ &\supset (A \,\mathbf{says}\, (\mathbf{goal}(URL, nonce))).\end{aligned}$$

3.9.2 Bundling-Based Relationships

We now incorporate bundling-based relationships into the logic. In particular, we introduce a new operator and an axiom exploiting this operator and prove soundness of the axiom by proving it as a theorem in the higher-order logic.

We add an operator $\mathbf{bundles}(A, URL_{personal}, URL_{location})$ expressing that principal A wants anyone that is granted an access right to $URL_{personal}$ also to be granted an access right to $URL_{location}$. The following axiom formalizes the application of this operator:

$$\begin{aligned}&\vdash (A \,\mathbf{says}\, (\mathbf{bundles}(A, URL_{personal}, URL_{location})))\\ &\wedge (A \,\mathbf{says}\, (\mathbf{delegate}(A, B, URL_{personal})))\\ &\supset (A \,\mathbf{says}\, (\mathbf{delegate}(A, B, URL_{location}))).\end{aligned} \tag{3.11}$$

An alternative option, as suggested by Bauer [14], is

$$\vdash (A\ \mathbf{says}\ (\mathbf{bundles}(A,\ URL_{personal},\ URL_{location})))$$
$$\wedge\ ((B\ \mathbf{says}\ (\mathbf{goal}(URL_{personal},\ nonce))) \supset (A\ \mathbf{says}\ (\mathbf{goal}(URL_{personal},\ nonce))))$$
$$\supset\ ((B\ \mathbf{says}\ (\mathbf{goal}(URL_{location},\ nonce))) \supset (A\ \mathbf{says}\ (\mathbf{goal}(URL_{location},\ nonce)))),$$

which expands the definition of the **delegate** operator. The first option seems to fail to grant a principal C access to $URL_{location}$ based on forwarded access rights, such as $A\ \mathbf{says}\ (\mathbf{delegate}(A, B,\ URL_{personal}))$ and $B\ \mathbf{says}\ (\mathbf{delegate}(B, C,\ URL_{personal}))$, since $A\ \mathbf{says}\ (\mathbf{bundles}(A,\ URL_{personal},\ URL_{location}))$ cannot be combined with the second statement. However, given the first two statements, we can prove the statement $A\ \mathbf{says}\ (\mathbf{delegate}(A, C,\ URL_{personal}))$, which can be combined with the third statement. Therefore, both options are equally expressive; we adopt the first one in this section.

Note that both options do not allow principal C to be granted access based on statements $A\ \mathbf{says}\ (\mathbf{delegate}(A, B,\ URL_{location}))$ and $B\ \mathbf{says}\ (\mathbf{delegate}(B, C,\ URL_{personal}))$ and the given information relationship. The difficulty is that the principal entitled to define information relationships for some information is not bound to this information (as it is the case in our information-representation scheme).

In Section 3.9.3, we will see that the **bundles** operator is a special case of the **combines** operator, which we will use for combination-based relationships. Therefore, we could define the **bundles** operator in terms of the **combines** operator and prove soundness only for this operator. However, for presentation reasons, we prove soundness also for the **bundles** operator. Proving soundness is easier for this operator and helps understand the more complex proof for the **combines** operator.

Using Bauer's notation, we define the **bundles** operator in the higher-order logic as follows :

$$\mathbf{bundles}(A,\ URL_{personal},\ URL_{location}) \stackrel{\text{def}}{=}$$
$$\forall B.(A\ \mathbf{says}\ (\mathbf{delegate}(A, B,\ URL_{personal})))$$
$$\rightarrow (A\ \mathbf{says}\ (\mathbf{delegate}(A, B,\ URL_{location}))).$$

Based on this definition, we can prove Axiom 3.11 as a theorem in the higher-order logic. We give this proof in Table 3.3. The proof uses P and L instead of $URL_{personal}$ and $URL_{location}$, respectively. It is based on Bauer's notation and refers to several theorems proved by Bauer [16, Chapter 3].

Table 3.3: Proof of Axiom 3.11

1	A **says** $(\mathbf{bundles}(A, P, L))$	premise
2	A **says** $(\mathbf{delegate}(A, B, P))$	premise
3	$[\forall B'.A$ **says** $(\mathbf{delegate}(A, B', P)) \rightarrow A$ **says** $(\mathbf{delegate}(A, B', L))]$	assumption
4	A **says** $(\mathbf{delegate}(A, B, P)) \rightarrow A$ **says** $(\mathbf{delegate}(A, B, L))$	$\forall_B$ e 3
5	$(\forall B'.A$ **says** $(\mathbf{delegate}(A, B', P)) \rightarrow A$ **says** $(\mathbf{delegate}(A, B', L)))$ $\rightarrow (A$ **says** $(\mathbf{delegate}(A, B, P)) \rightarrow A$ **says** $(\mathbf{delegate}(A, B, L)))$	$\rightarrow$ i 3-4
6	A **says** $\big((\forall B'.A$ **says** $(\mathbf{delegate}(A,B',P))\rightarrow A$ **says** $(\mathbf{delegate}(A,B',L)))$ $\rightarrow(A$ **says** $(\mathbf{delegate}(A,B,P))\rightarrow A$ **says** $(\mathbf{delegate}(A,B,L)))\big)$	SAYS-I2 5
7	A **says** $(\forall B'.A$ **says** $(\mathbf{delegate}(A, B', P))$ $\rightarrow A$ **says** $(\mathbf{delegate}(A, B', L)))$	$\stackrel{\text{def}}{=}$ e 1
8	A **says** $(A$ **says** $(\mathbf{delegate}(A, B, P))$ $\rightarrow A$ **says** $(\mathbf{delegate}(A, B, L))))$	SAYS-I3 6,7
9	A **says** $(A$ **says** $(\mathbf{delegate}(A, B, L)))$	SAYS-I3 2,8
10	A **says** $(\mathbf{delegate}(A, B, L))$	SAYS-TAUT 9

3.9.3 Combination-Based Relationships

To incorporate combination-based relationships into the logic, we introduce an operator $\mathbf{combines}(A, URL_{calendar}, B, URL_{location}, C, URL_{activity}, \ldots)$ expressing that principal A wants anyone who is granted an access right obtained from principal B to $URL_{location}$ and who has access rights obtained from principal C to $URL_{activity}$ (and who has potentially other access rights) to have access rights for $URL_{calendar}$. We formalize the application of this operator as follows:

$$\begin{aligned}
&\vdash (A \textbf{ says } (\mathbf{combines}(A, URL_{calendar}, B, URL_{location}, C, URL_{activity}, \ldots))) \\
&\wedge (B \textbf{ says } (\mathbf{delegate}(B, D, URL_{location}))) \\
&\wedge (C \textbf{ says } (\mathbf{delegate}(C, D, URL_{activity}))) \\
&\wedge \ldots \\
&\supset (A \textbf{ says } (\mathbf{delegate}(A, D, URL_{calendar}))). \qquad (3.12)
\end{aligned}$$

We define the **combines** operator in the higher-order logic as follows:

$$\begin{aligned}&\mathbf{combines}(A,\, URL_{calendar},\, B,\, URL_{location},\, C,\, URL_{activity},\, \ldots) \stackrel{\text{def}}{=} \\ &\quad \forall D.(B\ \mathbf{says}\ (\mathbf{delegate}(B, D,\, URL_{location}))) \\ &\quad \wedge\ (C\ \mathbf{says}\ (\mathbf{delegate}(C, D,\, URL_{activity}))) \\ &\quad \wedge \ldots \rightarrow (A\ \mathbf{says}\ (\mathbf{delegate}(A, D,\, URL_{calendar}))).\end{aligned}$$

Based on this definition, we can prove Axiom 3.12 as a theorem in the higher-order logic. We give this proof in Table 3.4. The proof uses CA, LO, and AC instead of $URL_{calendar}$, $URL_{location}$, and $URL_{activity}$, respectively. It refers to several theorems proved in Bauer [16, Chapter 3]. In particular, it refers to SAYS-I3', which is a generalized version of SAYS-I3 [16, p. 45], or

$$\begin{aligned}&\vdash (A\ \mathbf{says}\ (B \wedge C \rightarrow D)) \wedge (A\ \mathbf{says}\ B) \wedge (A\ \mathbf{says}\ C) \\ &\supset (A\ \mathbf{says}\ D). \qquad \text{(SAYS-I3')}\end{aligned}$$

Note that since the **says** operator is defined in terms of SAYS-I3, it needs to be redefined in terms of SAYS-I3'.

As mentioned before, the **bundles** operator is a special case of the **combines** operator. In particular,

$$\mathbf{bundles}(A,\, URL_{personal},\, URL_{location}) \equiv \mathbf{combines}(A,\, URL_{location},\, A,\, URL_{personal}).$$

3.9.4 Granularity-Based Relationships

It would also make sense to incorporate granularity awareness into Bauer's application-specific logic. Namely, if an entity had an access right to a URL, the entity should also have access rights to all the directories listed in the URL. However, since the logic is oblivious to the precise representation scheme of the information to which access is granted, we cannot formalize this concept in the presented application-specific logic.

3.9.5 Discussion

We have defined formal models for information relationship-aware access control in the context of both Lampson et al.'s theory of authentication (short: TOA) and Bauer's application-

Table 3.4: Proof of Axiom 3.12.

1	A **says** $(\mathbf{combines}(A, CA, B, LO, C, AC))$	premise
2	B **says** $(\mathbf{delegate}(B, D, LO))$	premise
3	C **says** $(\mathbf{delegate}(C, D, AC))$	premise
4	$[\forall D'.B$ **says** $(\mathbf{delegate}(B, D', LO)) \wedge C$ **says** $(\mathbf{delegate}(C, D', AC))$ $\rightarrow A$ **says** $(\mathbf{delegate}(A, D', CA))]$	assumption
5	B **says** $(\mathbf{delegate}(B, D, LO)) \wedge C$ **says** $(\mathbf{delegate}(C, D, AC))$ $\rightarrow A$ **says** $(\mathbf{delegate}(A, D, CA))$	$\forall_D$ e 3
6	$(\forall D'.B$ **says** $(\mathbf{delegate}(B,D',LO))\wedge C$ **says** $(\mathbf{delegate}(C,D',AC))\rightarrow A$ **says** $(\mathbf{delegate}(A,D',CA)))$ $\rightarrow(B$ **says** $(\mathbf{delegate}(B,D,LO))\wedge C$ **says** $(\mathbf{delegate}(C,D,AC))\rightarrow A$ **says** $(\mathbf{delegate}(A,D,CA)))$	i 4 5
7	A **says** $(\forall D'.B$ **says** $(\mathbf{delegate}(B,D',LO))\wedge C$ **says** $(\mathbf{delegate}(C,D',AC))\rightarrow A$ **says** $(\mathbf{delegate}(A,D',CA)))$ $\rightarrow(B$ **says** $(\mathbf{delegate}(B,D,LO))\wedge C$ **says** $(\mathbf{delegate}(C,D,AC))\rightarrow A$ **says** $(\mathbf{delegate}(A,D,CA)))$	SAYS-I2 6
8	A **says** $\big(\forall D'.B$ **says** $(\mathbf{delegate}(B,D',LO))\wedge C$ **says** $(\mathbf{delegate}(C,D',AC)))$ $\rightarrow A$ **says** $(\mathbf{delegate}(A,D',CA))\big)$	$\stackrel{\text{def}}{=}$ e 1
9	A **says** $\big(B$ **says** $(\mathbf{delegate}(B,D,LO))\wedge C$ **says** $(\mathbf{delegate}(C,D,AC))$ $\rightarrow A$ **says** $(\mathbf{delegate}(A,D,CA))\big)$	SAYS-I3 7,8
10	A **says** $(B$ **says** $(\mathbf{delegate}(B, D, LO)))$	SAYS-I2 2
11	A **says** $(C$ **says** $(\mathbf{delegate}(C, D, AC)))$	SAYS-I2 3
12	A **says** $(A$ **says** $(\mathbf{delegate}(A, D, CA)))$	SAYS-I3' 9-11
13	A **says** $(\mathbf{delegate}(A, D, CA))$	SAYS-TAUT 12

specific logic (short: ASL). Let us now elaborate on a few differences between the two formal models.

For TOA, we distinguish between the establishment of information relationships (Axiom (3.2)) and their application in access control (Axioms (3.3) and (3.4)). For ASL, we do not have this separation (Axioms (3.11) and (3.12)). We choose this approach since it is consistent with the different designs for dealing with speaks-for relationships in TOA and in ASL. In particular, the handoff axiom in TOA (Axiom (2.2)) defines the establishment of a speaks-for relationship between two principals, that is, the rule has a speaks-for operator in its conclusion. ASL has no such rule. Instead, as shown by Axiom (3.10), the speaks-for relationship is defined and applied in a single statement. Furthermore, differentiating between the definition of a relationship and its application in ASL would have required a method for identifying valid statements that establish relationships. In TOA, this identification is made easy by the inclusion of the owner of information in the information representation scheme. However, in ASL, the representation scheme does not include this owner.

For many purposes, not distinguishing between the establishment and application of a speaks-for relationship or an information relationship is sufficient. However, there are cases where the additional flexibility offered by differentiating between the two concepts proves useful. In terms of speaks-for relationships, TOA allows the establishment of $C \Rightarrow A$ out of $B \Rightarrow A$ and B **says** $(C \Rightarrow A)$, which is not possible for ASL. In this way, B can let C speak for A without having to give C all of B's access rights by letting C speak for B. To work around this problem, ASL provides the **delegate** operator, which allows B to transfer only a subset of its access rights to C. In terms of information relationships, TOA supports relationships that are defined by an entity different from the entity issuing an access right to the information on the right-hand side of the relationship, as demonstrated in the context of global information relationships, which is not possible for ASL. However, if necessary, we can introduce additional axioms into ASL which support more flexible speaks-for and information relationships.

We have proved soundness of the axioms that incorporate information relationships into ASL. We have not proved soundness of the corresponding axioms in TOA. However, the similarity between TOA's and ASL's axioms suggests that soundness could be proved if we defined TOA's axioms in Bauer's higher-order logic.

Incorporating a new formal axiom into our access-control architecture requires the definition of a Java class that encapsulates the axiom, as discussed in Section 2.6.3. Furthermore, the proof-building algorithm needs to be modified to take the new axiom into account. In summary, incorporating a new axiom requires source-code changes. An advantage of ASL is that ASL does not require source-code changes, since it uses the Twelf

logical framework [104] for building and verifying proofs of access. In terms of proof building, an axiom is represented in the prover directly in its logical form. The introduction of a new axiom simply requires the addition of a corresponding tactics to the prover [16, p. 61]. In terms of proof verification, ASL employs a preprocessor that rejects illegal axioms, replaces digital signatures with a special axiom, and ensures that the proof tries to prove the expected statement. This preprocessor needs to be made aware of a newly introduced axiom. After the preprocessing step, ASL has the proof verified by Twelf. This verification step does not require any changes for a new axiom.

3.10 Related Work

There are several areas of research that are relevant to our work. As mentioned in Section 1.2.2, pervasive computing projects that focus on controlling access to information [38, 51, 66] have not studied access control to complex information. Let us review some other related work, namely, in the areas of bundling information, threshold constructs, and information derivation.

The concept of bundling information has been used in environments other than pervasive computing. For example, individuals store files belonging to the same project in the same directory or they keep pictures from a trip in the same gallery. In the context of object-oriented databases, Rabitti et al. [106, 19] observe that objects stored in a centralized database can be organized in a hierarchical fashion, showing how an object is composed of objects at successively lower levels. The authors exploit this observation to assign (positive or negative) access rights to an object. Objects at a lower level inherit an access right, unless they are explicitly assigned a different access right. Our bundling-based relationships have the advantage that they can be established in a distributed way. Similar to Rabitti et al., Jajodia et al. [71] present several possibilities for inheriting access rights by organizing subjects, resources, and types of access rights in hierarchies. Their model applies only to centralized environments.

Combination-based relationships are related to threshold constructs in trust-management systems, such as SPKI/SDSI [47], KeyNote [22], or RT [88]. These constructs require that multiple principals issue an access right to a principal before this principal can access information. For satisfying this requirement, all access rights issued by the various principals must grant access to the *same* information. Therefore, we cannot use threshold constructs for controlling access to complex information. Namely, a principal should be able to access complex information only if the principal has access rights to *different* kinds of information, potentially issued by different principals.

Atluri and Gal [9] examine a centralized "information portal", in which information items are derived from base information items. An administrator can grant explicit access rights to items. In addition, if a principal has access rights to all base information items used for deriving a new item, the principal will also have access to the derived item. This concept sounds similar to combination-based relationships. However, the motivations for the two concepts are different. In the earlier work, information is derived from other information, therefore, access rights to the new information depend on access rights to the base information. In our work, complex information reveals other information, but is not actually derived from this information. Combination-based relationships have the advantage that they do not assume a centralized environment, where an administrator defines how information is derived.

Similar to Atluri and Gal, Myers and Liskov [95] study information confidentiality in an information derivation scenario. Their approach is based on flow control, and they label information with its access policy. If new information is derived from base information, the label of the new information will be determined from the labels of the base information such that only entities being able to access all the base information have access to the derived information. Myers and Liskov perform their research in the context of downloaded program code, which needs to be prevented from leaking information.

3.11 Summary

In this chapter, we showed how to take the semantics of complex information into account for access control in pervasive computing. In particular, we capture the semantics with information relationships. Relationships also decrease the number of access rights than an individual needs to issue. We proposed making such relationships first-class citizens in access control, identified three types of information relationships that are important in pervasive computing, formalized their establishment and application in access control, and integrated support for them in a client-based access-control architecture.

Our sample implementation and its deployment demonstrate the feasibility of our approach. Its performance is competitive. Based on the complexity analysis of proof building and its measurement-based validation, we anticipate small cost for building proofs in future pervasive computing environments.

For complex information, its owner needs to decide whether to establish a combination-based relationship for the information. The owner can also issue explicit access rights to the information. For other types of information, the owner has to decide whether to define bundling-based information relationships for the information. Whereas both combination-

based and bundling-based information relationships can reduce the number of actual statements that an owner of information needs to make, these relationships also increase the set of options that our access-control architecture makes available to the owner. The availability of more options does not necessarily lead to an improvement in terms of usability of our architecture. In particular, in order to exploit the new options, an owner of information needs to be aware of them and know how they work. We expect the new options to be intuitive. However, only a large-scale deployment of information relationships can validate this expectation.

In the next chapter, we examine how we can make services that derive information more secure in case of an intrusion.

Chapter 4

Derivation-Constrained Access Control

Access control ensures that only authorized clients get access to confidential information provided by a service. Often, when a client issues a request for information provided a service, this service acts as a gateway and issues further requests for information to other services, acting as endpoints. For example, a people-location service can issue a request for the location of a device when receiving a request for the location of a person. A service providing health information might issue queries for a person's medical information (heart rate, blood pressure,..), as delivered by wearable sensors, when receiving a query for a person's health. There could also be multiple gateways between the client and the endpoint(s).

This level of indirection makes access control difficult. On the one hand, a gateway needs access rights in order to retrieve information from other services. On the other hand, granting access rights to a gateway makes the gateway vulnerable in case of an intrusion. Namely, an intruder into the gateway has access to any information that the gateway is authorized to get, and the intruder can actively issue requests for information to other services. Services will answer these requests if the gateway is authorized to access the requested information.

The conventional solution to this problem exploits short-lived access rights. In particular, when a client issues a request for information to a gateway, the gateway is granted short-lived access rights, which allow the gateway to temporarily retrieve the information required for answering the client's request from a service. A drawback of this solution is its lack of flexibility. The solution is easy to implement if the entity granting access to the gateway and the client asking for information are identical, but it becomes more difficult if they are not. One of the difficulties is that the entity granting access might be offline upon a request and thus not be able to issue access rights.

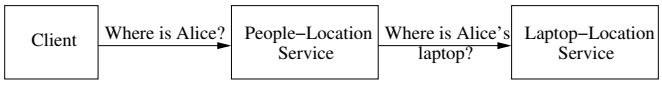


Figure 4.1: Gateway. The people-location service acts as a gateway between the client and the laptop-location service.

In this chapter, we propose *derivation-constrained access control*, where an endpoint, before returning information to a gateway, requires the gateway to prove that the gateway is asking for this information in order to answer an authorized request for derived information sent to the gateway by a client. If there is no such request, an intruder into the gateway will not be able to prove that the gateway is asking for the information on a client's behalf. Therefore, the endpoint will not grant the gateway (and the intruder) access to the requested information.

4.1 Overview

Let us illustrate derivation-constrained access control with an example application, which we will use throughout this chapter. There is a service that locates people and a service that locates laptops (see Figure 4.1). The latter service locates people indirectly by locating their laptops, that is, the people-location service acts as a gateway and contacts the laptop-location service upon a client's request for a person's location. Assume that Bob asks the people-location service for Alice's location information. The people-location service then asks the laptop-location service for the location of Alice's laptop. Using derivation-constrained access control, the laptop-location service now makes the people-location service prove that the people-location service is issuing this request in order to answer Bob's authorized request for Alice's location and that the location of Alice's laptop should be used for deriving Alice's location.

Derivation-constrained access control does not suffer from the drawback that we observed for a solution based on short-lived access rights. Our approach requires the entity that grants access to the information provided by an endpoint to specify what information is derived from this information. However, there is no need to issue these specifications upon a request; they need to be issued only when there is a change in the derivation properties of information. Therefore, the entity does not need to be online when a request is made.

For this chapter, we extend the security model introduced in Section 2.7. We assume

that gateways are not malicious to start with, but that they become corrupted if an attacker breaks into them. In the case of corruption, an attacker has full control over a gateway and uses the gateway to contact other services. We assume that clients that are authorized to get information offered by a gateway and corrupted gateways do not collude. Similarly, we assume that services and corrupted gateways do not collude.

The contributions of this chapter are

- the concept of exploiting derivation properties of information as a first-class citizen in a client-based access-control architecture,
- a formal model for incorporating derivation properties into access control,
- a prototype implementation based on this model, and an evaluation of this implementation.

We start by discussing some background material (Section 4.2). We introduce derivation-constrained access control (Section 4.3) and incorporate it into our formal model (Section 4.4). We apply the formal model to two different example scenarios (Section 4.5) and present a security analysis (Section 4.6). Finally, we discuss a prototype implementation (Section 4.7) and analyze its performance (Section 4.8).

4.2 Background

In this section, we discuss several application scenarios for gateways and contrast the approaches chosen in related work with derivation-constrained access control.

There are two types of gateways. The first type of gateways has the property that an authorized client has access rights both to the information offered by a gateway and to the information provided by an endpoint. For example, Alice has access to both her raw medical information, as collected by individual information services (i.e., sensors), and her health status, as derived by a gateway. In another example, she can access both her calendar information, as provided by a calendar application, and her location information, as derived from her calendar information by a gateway.

There has been a lot of work on this type of gateways [53, 69, 78, 81, 97, 115]. Because a client has access rights to the information offered by a service, this work relies on delegation chains and lets the client delegate its access rights to a gateway. In the rest of this paper, we call this kind of gateways *delegation gateways*.

The second type of gateways has the property that an authorized client has access rights to the information offered by a gateway, but the client does not have access rights to the information offered by an endpoint. For example, a face detection application acts as a gateway and extracts somebody's location from a videostream. Though Alice's location can be derived from a videostream and Alice can access her location information, she cannot access the videostream directly. In another example, Alice might have access to Bob's location information, but she does not have access to Bob's calendar, from which a gateway derives his location. Finally, a gateway can derive the location of a person from the location of her laptop. A company might decide that, for administrative reasons, its employees should not have access rights to their laptop's location information. On the other hand, an employee should have access rights to her location information, which can be derived from her laptop's location information.

Related work has largely ignored the second type of gateways. Delegation chains are not applicable here, since the client does not have access rights to the information offered by an endpoint. Instead, derivation gateways are typically assumed to have separate access rights to information provided by an endpoint. Therefore, access control ignores that this information is used by a gateway for deriving other information, which makes the gateway vulnerable in case of an intrusion. In the rest of this paper, we call this kind of gateways *derivation gateways*. Derivation-constrained access control exploits derivation properties of information and makes a gateway prove to an endpoint that the gateway is asking for information in order to answer a client's authorized request for derived information. Derivation-constrained access control has the additional benefit that a gateway does not have to make an access decision. Only endpoints need to make such decisions. Therefore, we can run access control in an end-to-end way.

We introduce support for derivation gateways by formalizing derivation properties of information. With the help of these properties, we can tie a gateway's request for information to a client's request for derived information and exploit this knowledge in access control. There is no need to specify derivation properties for each request, they need to be defined only when there is a change in the derivation properties of information.

Derivation-constrained access control also supports delegation gateways. Previous work that addresses the problem of gateway corruption for delegation gateways [53, 78, 81] has a client delegate short-lived access rights to a gateway. Derivation-constrained access control follows the same approach. Compared with previous work, its advantage is that it allows a client to hand over only the specific set of access rights required for answering the client's request to a gateway, not all of its access rights.

4.3 Derivation-Constrained Access Control

We now present the design of derivation-constrained access control. Since delegation gateways are a subset of derivation gateways, we focus our description on derivation gateways. In Section 4.5.2, we present an example demonstrating the usage of derivation-constrained access control for delegation gateways.

Derivation-constrained access control requires two basic components. First, we need *derivation-constrained access rights*, which constrain access rights to an entity such that the entity has to prove that it is asking for the information listed in the access right in order to answer an authorized request for derived information. Second, in order to be able to identify derived information, we need formal *derivation properties* between information a gateway asks for and information a client asks for.

Instead of having two separate components, it is possible to directly include derivation properties in access rights. This approach has the drawback that it increases the number of access rights that need to be issued. However, it turns out that the approach restricts the flexibility of intruders that have broken into a gateway. We discuss this trade-off in Section 4.6.

We now examine access rights and derivation properties in more detail and describe their application in derivation-constrained access control.

4.3.1 Derivation-Constrained Access Rights

As mentioned in Section 2.2.2, an access right can have constraints associated with it. In derivation-constrained access control, a constraint requires that the information in the access right is used for answering a request for derived information, as discussed in Section 4.1. We call access rights having such a constraint *derivation-constrained access rights*.

In our example application, Alice's company, as the owner of Alice's laptop, defines a derivation-constrained access right stating that the people-location service can access the location of Alice's laptop.

4.3.2 Derivation Properties

We need derivation properties to tie information asked for by a gateway to information asked for by a client. In particular, a derivation property states that *issuer information*

is derived from *subject information*. Issuer information is the information offered by a service. A gateway uses the issuer information to derive the subject information.

We require that only the owner of the issuer information can specify a valid derivation property for this information. The reason for this requirement is that a derivation property is used for controlling access to the issuer information. If a random entity was able to define derivation properties for the issuer information, the client could define a derivation property having information that the client owns as subject information. Therefore, the client would be able to send an authorized request for this subject information to a gateway. Assuming the gateway is granted access to the issuer information in a (derivation-constrained) access right, the gateway could then successfully query a service for this issuer information and leak the (derived) information to the client.

The same issuer information can show up in multiple derivation properties. For example, Alice's current calendar entry could be used for deriving both her current location and her current activity. Similarly, subject information can occur in multiple derivation properties. For example, Alice's current location could be derived from her current calendar entry or from her laptop's current location.

Let us revisit our example application. Alice's company, as the owner of Alice's laptop, defines a derivation property having the laptop's location information as issuer information and Alice's location information as subject information.

4.3.3 Access Control

We now discuss how derivation-constrained access rights and derivation properties are used in client-based access control.

When sending a request for information to an endpoint, a gateway checks whether there are access rights that grant the gateway access to the requested information. The gateway first tries to exploit access rights that are not derivation-constrained. If there is no such access right, the gateway tries to exploit derivation-constrained access rights and derivation properties. In this case, it needs to assemble a proof of access for the following facts:

- There must be one or a chain of derivation-constrained access rights that grant access to the requesting gateway.
- There must be a derivation property having the requested information as issuer information.

- There must be a request from a client to the gateway asking for the subject information in the derivation property.
- There must be one or a chain of access rights that grant the client access to this subject information.
- To avoid replay attacks, the client's request needs to be fresh, that is, the service must not have seen it before.

In our example application, the people-location service requires both Alice's derivation-constrained access right and the corresponding derivation property. In addition, it needs a fresh request from, for example, Bob asking for Alice's location information and an access right granting Bob access to this information.

After having introduced derivation-constrained access control informally, we incorporate it into our formal model in the next section.

4.4 Formal Model

In this section, we introduce a notation for defining derivation-constrained access rights and derivation properties and discuss their application in access control.

As explained in Section 2.6.2, Abadi et al. [1] introduce the construct D **controls** t to encode that principal D is on a service's ACL for resource t. The construct is defined as

$$D \textbf{ controls } t \equiv ((D \textbf{ says } t) \supset t).$$

For our purposes, this definition is not sufficient. It is important to guarantee freshness of a request in order to avoid replay attacks, as stated in Section 4.3.3. Strictly speaking, freshness demands that a request is new. We relax this requirement and demand only that a request is recent. As it turns out, our formal model easily supports this relaxed requirement, and, as we explain in Section 4.7, the original requirement is difficult to achieve in our problem setting.

To formalize freshness, we observe that Lampson et al. [81] suggest a statement form t **until** $\mathcal{T}$ to denote the lifetime $\mathcal{T}$ of statement t. We consider a statement fresh if its lifetime has not expired and extend Abadi et al.'s definition of access control to support freshness of a request: ($\mathcal{T}_{now}$ is the current time.)

$$D \textbf{ controls } t \equiv ((D \textbf{ says } t) \textbf{ until } \mathcal{T}) \supset t \quad \text{if } \mathcal{T} \leq \mathcal{T}_{now}. \tag{4.1}$$

We are now ready to formalize derivation-constrained access control, as discussed in Section 4.3, by extending the definition of D **controls** t .

We first introduce the convention to mark information in derivation-constrained access rights with the + sign (e.g., A **says** $(B \overset{D.x^+}{\Longrightarrow} A)$. In this way, when a service detects marked information in a speaks-for statement, it concludes that this statement should be used only for derivation-constrained access control, not for conventional access control.

Intuitively, having an access right to information $D.x$ should imply having an access right to information $D.x^+$. In terms of information relationships, we can formulate this requirement as $D.x \longrightarrow D.x^+$. This relationship allows us to combine speaks-for statements for $D.x$ and $D.x^+$ based on Axiom (3.3), or $\vdash (C \overset{D.x^+}{\Longrightarrow} B) \wedge (B \overset{D.x}{\Longrightarrow} A) \supset (C \overset{D.x^+}{\Longrightarrow} A)$.

We also introduce a notation for expressing derivation properties. The statement

$$E.y \mapsto D.x \tag{4.2}$$

denotes that subject information $E.y$ can be derived from issuer information $D.x$. For example, $D.x$ could be the location information of Alice's laptop, as owned by her company, and $E.y$ could be Alice's location information, as owned by Alice.

As explained in Section 4.3.2, we assume that for a derivation property to hold, the principal owning the issuer information or a principal speaking for that principal needs to issue the statement, that is,

$$\vdash (F \textbf{ says } (E.y \mapsto D.x)) \wedge (F \overset{D.x}{\Longrightarrow} D) \supset (E.y \mapsto D.x). \tag{4.3}$$

We can now formulate derivation-constrained access control. In particular, we have a service use a different definition for D **controls** t if t involves information marked with the + sign. Formally,

$$
\begin{aligned}
& D \textbf{ controls } \text{“read } D.x^{+}\text{”} \\
& \equiv ((D \textbf{ says } \text{“read } D.x^{+}\text{”}) \textbf{ until } \mathcal{T}_1 \\
& \wedge E.y \mapsto D.x \\
& \wedge (E \textbf{ says } \text{“read } E.y\text{”}) \textbf{ until } \mathcal{T}_2) \\
& \supset \text{“read } D.x^{+}\text{”} \quad \text{if } \mathcal{T}_1 \leq \mathcal{T}_{now} \text{ and } \mathcal{T}_2 \leq \mathcal{T}_{now}.
\end{aligned}
\tag{4.4}
$$

The first condition is straightforward: There must be a fresh and authorized request, “read $D.x^{+}$”. In the second condition, we require that there is information $E.y$ that can be derived from $D.x$.[1] The third condition calls for a fresh and authorized request for the derived information, $E.y$.

Note that the third condition can be replaced by a condition requiring a request for $E.y^{+}$ and conditions requiring the existence of another derivation property and of another request. This property makes deployment of our mechanism feasible in scenarios where a client’s request for information is dealt with by multiple, cascaded gateways between the client and the endpoint.

Derivation-constrained access control makes proofs of access more complex. It is no longer sufficient for a gateway to prove that it can speak for the owner of the requested information regarding this information, as outlined in Section 2.6.2. Instead, Definition (4.4) requires separate speaks-for relationships for the first and third condition and a valid derivation property. The two speaks-for relationships demonstrate that the gateway can speak for the issuer information and the subject information in the derivation property, respectively. We demonstrate such a proof of access in the next section.

4.5 Examples

In this section, we demonstrate the application of derivation-constrained access control in two example scenarios.

4.5.1 People Location

We present an example of a derivation gateway. In our scenario, a people-location service (PL) derives Alice’s location information (i.e., Alice.location) from her laptop’s location

[1] It is possible to associate a lifetime with the property, we leave it away for readability reasons.

Table 4.1: Statements in the people-location scenario. (S1)-(S4) are established during setup, (S5)-(S16) for access control.

Step	Statements
(S1)	(Alice **says** Bob $\overset{\text{Alice.location}}{\Longrightarrow}$ Alice) **until** $\mathcal{T}_1$
(S2)	(ACME **says** PL $\overset{\text{(ACME, Alice\_laptop).location}^+}{\Longrightarrow}$ ACME) **until** $\mathcal{T}_2$
(S3)	ACME **says** Alice.location $\mapsto$ (ACME, Alice_laptop).location
(S4)	ACME **controls** (ACME, Alice_laptop).location$^+$
(S5)	Bob **says** "read Alice.location"
(S6)	(Bob $\overset{\text{Alice.location}}{\Longrightarrow}$ Alice) **until** $\mathcal{T}_1$
(S7)	(Bob **says** PL|Bob $\overset{\text{Alice.location}}{\Longrightarrow}$ Bob) **until** $\mathcal{T}_3$
(S8)	(PL $\overset{\text{(ACME, Alice\_laptop).location}^+}{\Longrightarrow}$ ACME) **until** $\mathcal{T}_2$
(S9)	PL **says** "read (ACME, Alice_laptop).location$^+$"
(S10)	(ACME **says** "read (ACME, Alice_laptop).location$^+$") **until** $\mathcal{T}_2$
(S11)	Alice.location $\mapsto$ (ACME, Alice_laptop).location
(S12)	(PL|Bob $\overset{\text{Alice.location}}{\Longrightarrow}$ Bob) **until** $\mathcal{T}_3$
(S13)	(PL|Bob $\overset{\text{Alice.location}}{\Longrightarrow}$ Alice) **until** $\mathcal{T}_3$
(S14)	PL|Bob **says** "read Alice.location"
(S15)	Alice **says** "read Alice.location" **until** $\mathcal{T}_3$
(S16)	"read (ACME, Alice_laptop).location$^+$"

information (i.e., (ACME, Alice_laptop).location), where the laptop is owned by Alice's company, ACME, and its location information is offered by a laptop-location service. Note that the two types of information are owned by two different entities. We show the relevant statements in Table 4.1.

We start with statements that individuals make before an actual request is issued. Alice grants Bob access to her location information (S1) and gives the corresponding statement to Bob. We expect this statement to be long lived. ACME grants the people-location service access to Alice's laptop's location information in a derivation-constrained access right (S2), again in a long-lived statement, and gives the statement to the service. ACME also states that the laptop's location information can be used for deriving Alice's location information (S3) and hands over the statement to the people-location service. Finally, the device location service has ACME on its ACL for the laptop's location information (S4).

Bob sends a request for Alice's location information to the people-location service (S5).

His proof of access includes a speaks-for statement (S6) derived from the access right that Bob received from Alice (S1). Bob also issues a short-lived access right for Alice's location information to the people-location service (S7) and sends this access right to the people-location service. In order to use this access right, the people-location service must quote Bob (denoted by "|"). As observed by Howell and Kotz [69], this quoting prevents a gateway from having to make an access decision itself and allows an endpoint to perform end-to-end access control instead. In particular, quoting makes it possible for an endpoint to identify the client making the original request to the gateway.

The people-location service now assembles a proof of access that lets it access information (ACME, Alice_laptop).location^{+}. In particular, the proof needs to show that each of the conditions in Definition (4.4) is fulfilled.

For the first condition, based on its access right (S2), the service proves that it speaks for ACME on behalf of Alice's laptop (S8). Therefore, ACME supports the request of the people-location service (S9) for the laptop's location information (S10).

For the second condition, the people-location service validates the derivation property (S3) using Axiom (4.3) and proves its validity (S11).

For the third condition, the people-location service derives a speaks-for statement (S12) from the access right received from Bob (S7). In addition, the service exploits the proof of access received from Bob showing that Bob can speak for Alice regarding her location information (S6). In summary, the people-location service (quoting Bob) can speak for Alice regarding her location information (S13). Therefore, when the service issues a (quoted) request for Alice's location (S14), Alice supports this request (S15). (We assume that $\mathcal{T}_3 << \mathcal{T}_1$ since Statement (S1) is long lived, whereas Statement (S7) is short-lived.)

The people-location service then sends statements (S8), (S11), and (S13), together with requests (S9) and (S14), to the device location service, which validates the statements and combines them with the requests and the service's ACL based on Definition (4.4) to conclude that the people-location service should be granted access (S16) (assuming $\mathcal{T}_2 \leq \mathcal{T}_{now}$ and $\mathcal{T}_3 \leq \mathcal{T}_{now}$).

4.5.2 Health Information

Let us look at the application of derivation-constrained access control to delegation gateways, where a client also has access rights to the information offered by an endpoint. To support delegation gateways, a client can delegate its access rights to the information provided by an endpoint to a gateway, potentially in a short-lived way. In related work [53, 78, 81, 97, 115], a client delegates all its access rights to information offered

by the endpoint to the gateway. Derivation-constrained access control supports more fine-grained delegation.

Let us study the example of deriving a person's health status from her medical information. For example, there is a derivation property

$$\text{Alice } \mathbf{says} \text{ Alice.health\_status} \mapsto \text{Alice.heart\_rate},$$

which a gateway uses to access information Alice.heart_rate at a service. In order to be granted access, the gateway requires (long-lived) derivation-constrained access to information Alice.heart_rate and (short-lived) access to information Alice.health_status. Alice has to delegate only these two access rights to the gateway, and only one of them for each request. Other access rights that Alice might have will not be delegated to the gateway.

4.6 Security Analysis

In this section, we analyze the security of derivation-constrained access control.

Since our formal model is based on Lampson et al.'s model, we inherit its properties. For example, the model does not prevent a principal having an access right from forwarding this access right to other principals. The reasoning is that if the model did not allow this forwarding, the principal could become a proxy for the other principals and access information on their behalf. For derivation-constrained access control, this principle implies that an intruder breaking into a gateway can forward the gateway's (derivation-constrained) access rights to other, potentially corrupted gateways. While this attack will not grant the first gateway access to additional information, the other gateways could exploit the forwarded access rights if there was an authorized request from a client to them.

As mentioned in Section 4.3, we keep access rights and derivation properties separate. A disadvantage of this approach is that intruders into a gateway get more flexibility. Assume that there are multiple derivation properties, all of them having the same issuer information. For example, multiple kinds of information can be derived from Alice's calendar. Alice grants derivation-constrained access rights to a gateway because Alice expects the gateway to provide one particular derivation. (For example, Alice expects the gateway to provide location information.) If an authorized client now asks the gateway for subject information listed in any of the derivation properties, the gateway will be granted access, even though Alice does not expect the gateway to perform the derivation listed in the derivation property exploited by the gateway. Therefore, an intruder into a gateway

can increase its chances of success by trying to trick authorized clients into contacting the corrupted gateway for any of the different types of subject information. We can look at this problem as an instance of the confused-deputy problem [58].

We can address this attack in different ways. At a low level, clients must use authentication to ensure that they really talk to the gateway owning a particular public key. At a high level, clients need to ensure that they obtain a gateway's genuine public key (or more general, genuine contact information) and not a fake key (fake contact information) produced by a malicious entity. For example, Alice could tell Bob which gateway he should use for obtaining her location information. At the formal level, we could address this attack by combining access rights and derivation properties. For example, $B \xRightarrow{\{\text{“}E.y \mapsto D.x\text{”}\}} A$ could be used to indicate that B can speak for A regarding "read $D.x$", but only if there is an authorized request for "read $E.y$". In this scenario, if B issues a request for $D.x$, but presents a request for derived information other than $E.y$, the request will be denied. However, this approach could require more access rights. Namely, if a gateway extracted multiple information items from the issuer information, it will need separate access rights for each information item. When derivation properties are separate, the service requires only one access right.

Derivation-constrained access control also fails when an owner of information issues a derivation property that has information owned by a corrupted gateway as subject information. In this scenario, the intruder into the gateway can grant itself access rights to this information and then issue a request for the subject information to the gateway. Since the gateway owns the subject information, a service will honor the gateway's requests for the issuer information listed in the derivation property (assuming that the gateway has constrained access to this issuer information). Therefore, owners of information need to exercise care when deciding what representation scheme for information they use upon issuing derivation properties.

When granting a service that derives information access to input information, the owner of this input information needs to choose between issuing an unconstrained and a derivation-constrained access right. Whenever possible, the owner should define derivation-constrained access rights since they make the service less susceptible in case of an intrusion. However, a derivation-constrained access right is not always sufficient. For example, a tracking service by design has to contact a location service continuously, even when there is no authorized request that is sent by a client to the tracking service. For such a service, the owner of the location information needs to issue an unconstrained access right. In this case, it might be possible to use other techniques to shield the input information from an intruder. For example, a service could compute on encrypted input information (see Section 4.9).

```
(derivation
   (version ''1'')
   (issuer
      (information (pub_key:acme) alice_laptop location))
   (subject
      (information (pub_key:alice) alice location)))


(cert
   (version ''1'')
   (issuer (pub_key:acme))
   (subject (pub_key:people_locator))
   (conditional)
   (permission
      (information (pub_key:acme) alice_laptop location))
   (tag *))
```

Figure 4.2: Derivation property and derivation-constrained access right. Alice's location information is derived from her laptop's location information. ACME grants the people-location service derivation-constrained access to the laptop's location information. (The digital signatures are omitted.)

4.7 Implementation

Our implementation is based on the client-based access-control framework introduced in Section 2.6, to which we added support for building proofs of access involving derivation-constrained access control, as outlined in Sections 4.4 and 4.5.

We use extended SPKI/SDSI certificates for expressing and storing derivation properties and derivation-constrained access rights. We give two examples in Figure 4.2. The first certificate states that Alice's location information, as owned by Alice, is derived from her laptop's location information, as owned by ACME. The second certificate says that ACME grants the people-location service an access right to the laptop's location information. We mark derivation-constrained access rights with a **conditional** entry.

As shown in Section 4.5.1, a client that issues a request to a gateway also needs to generate a short-lived access right that grants the gateway access to the information in the client's request. One way to express such an access right is to have the client issue a digital certificate. However, if there are multiple, cascaded gateways, each will have to

forward the access right to the next gateway in the chain and each will have to issue a digital certificate. Generating a certificate is expensive. In our implementation, we reduce cost by not using separate digital certificates for forwarding access rights to gateways. Instead, we have the original issuer of a request sign this request, where the lifetime of this signature is short. Gateways now forward access rights by forwarding this signed request. In the example described in Section 4.5, instead of issuing a digital certificate that grants the people-location service short-lived access to Alice's location information, Bob signs his request for this information before sending the request to the people-location service. We assume that any entity that is in the possession of a signed request is implicitly granted access to the information in the request. Formally, this entity can speak for the issuer of the signed request on behalf of the information in the request.

To be secure, this optimization requires two precautions: First, we have to ensure that attackers cannot snoop traffic and obtain access rights by observing signed requests. Therefore, communication has to be confidential. Confidentiality is required anyway because we exchange private information (such as a person's location information). Second, an endpoint can use a speaks-for statement that is implicitly derived from a signed request only for running access control for this particular request; it must never re-use such a speaks-for statement for other requests. The latter precaution is required since gateways do not make access decisions. Without it, an attacker could contact a gateway that previously obtained information from a service and have it contact the service again. If the service re-used the previously derived speaks-for statement, the statement would grant the gateway access again (unless the statement had expired).

To support the **until** construct, we exploit the fact that the lifetime of SPKI/SDSI certificates can be limited and have the issuer of a statement assign a lifetime to the statement. We use lifetimes to achieve freshness. We could also integrate a nonce-based approach, where the entity that wants to issue a request to a target receives a nonce from the target and returns this nonce in its request (as deployed by, for example, Bauer et al. [16] and described in Section 3.9.1). However, our communication pattern does not allow for nonces. In particular, a client sends its request to a gateway before the gateway sends a request to a service and gets the chance to obtain a nonce from the service.

4.8 Performance Analysis

We present a measurement-based analysis of derivation-constrained access control and discuss the results.

Table 4.2: Client-response time. Mean and standard deviation of elapsed time for security-related operations (in bold) and other expensive operations [ms].

Entity	Step	μ	(σ)
Client	**Request generation**	27	(3)
Client	**SSL sockets creation**	52	(4)
Gateway	**Access control**	2	(1)
Gateway	Gather badge information	26	(3)
Gateway	**Request generation**	41	(3)
Gateway	**SSL sockets creation**	52	(5)
Endpoint	**Access control**	3	(2)
Endpoint	Gather location information	49	(30)
	Total	310	(37)

4.8.1 Measurements

We measure the time it takes for a gateway and an endpoint to process a request, where processing includes access control. Our scenario roughly corresponds to the example application discussed in Section 4.5. We assume that Alice grants Bob access to her location information. Bob contacts a service providing people-location information. This service acts as a gateway and contacts a service that provides location information about badges worn by people. Alice's company, as the owner of the badge, specifies a derivation property between the badge's location information and Alice's location information. In addition, the company issues a derivation-constrained access access right that grants the people-location service access to the badge's location information.

Our experiments run on an unloaded Pentium IV/2.5 GHz with 1.5 GB of memory, Linux 2.4.20, and Java 1.4.2. The public and private key operations operate on RSA keys with a size of 1024 bit. All experiments are run 100 times.

We run the client, the people-location service, and the badge location service on the unloaded host. In addition, there is another host that runs a database storing digital certificates and static information about users and devices (e.g., ownership of devices and location of sensors detecting badges). The client and the location services contact this database to retrieve any data required for generating a request for information or processing such a request. Finally, there is a host that runs a commercial, Windows-based software application for locating RF/IR badges [123].

It takes about 310 ms for the client to send a request to the gateway and to receive a response. We present more detailed results in Table 4.2. The terms “client”, “gateway”, and “endpoint” represent Bob, the people-location service, and the badge location service, respectively. The most expensive operations are the two SSL sockets creations, one pair for the client-gateway connection and another pair for the gateway-endpoint connection. For opening an SSL connection, the two peers need to authenticate each other. Each authentication requires an RSA signing or decryption operation operation, which takes about 17 ms. RSA is also responsible for most of the cost of generating a request, because a request needs to be signed. In our current implementation, any entity issuing a request needs to sign the request. This approach supports scenarios, where there are multiple, cascaded gateways. If an entity knew that it is directly contacting an endpoint, it could omit this signature. For example, the people-location service would not have to sign the request for the badge’s location information sent to the badge location service, since this service is an endpoint. However, Bob needs to sign his request for Alice’s location information sent to the people-location service.

Compared to the cost for creating SSL sockets and generating a request, access control is inexpensive. The main cost is caused by signature verification, which takes about 1 ms per signature. The gateway validates the signatures of the client’s request and of the access right granting access to the client. Strictly speaking, derivation-constrained access control does not require a gateway to make an access decision. However, we have the gateway make such decisions, since the decisions are cheap and can help against denial-of-service attacks, where clients send unauthorized requests to a gateway. The endpoint needs to make access decisions. It validates the signatures of the client’s request, of the access right granting access to the client, of the access right granting derivation-constrained access to the gateway, and of the derivation property. Overall, it has to validate four signatures.

The two most expensive operations not related to access control are the gathering of badge information, as performed by the gateway, and the gathering of location information, as performed by the endpoint. For the former operation, the gateway contacts the external database in order to learn about Alice’s badge. For the latter operation, the endpoint contacts the external host for retrieving the badge’s location.

SSL allows two communicating entities to resume previous sessions. This feature avoids the two expensive RSA signing/decryption operations required for setting up a fresh SSL session. With session resume, we can reduce the cost of setting up an SSL socket to 10 ms (standard deviation: 3 ms) and the overall mean response time by 28% to 224 ms (48 ms). Session resume assumes that the same pair of entities communicate repeatedly in a time frame. Its usefulness strongly depends on the deployment environment. For example, the people-location service will contact the badge location service multiple

times (though not necessarily for the same device). A tracking application will contact the people-location service often, whereas a human might contact it only occasionally.

Finally, we note that the RSA operations are CPU-bound and benefit strongly from faster hardware and implementation optimizations. For example, on our evaluation machine, OpenSSL [105] completes an RSA signing operation (1024 bits) in 5 ms. A verification operation takes 0.3 ms.

4.8.2 Discussion

We now compare the cost of running derivation-constrained access control with the expected cost of alternative approaches. For delegation gateways, an alternative approach is to have a client delegate its access rights to the gateway in a short-lived way. The endpoint then needs to validate two digital signatures, one for the client's access right and one for the gateway's access right. In our approach, four signatures have to be checked. However, validating signatures is relatively cheap, and our approach supports fine-grained delegation.

For derivation gateways, an alternative approach is to have the owner of the information provided by the endpoint issue a long-lived access right to a gateway. This approach is less expensive than our approach. There is no need for the client to sign its request to the gateway (i.e., issue a short-lived access right to the gateway) and for the endpoint to validate this access right, the access right of the client, and the derivation property. However, long-lived access rights make the gateway vulnerable in case of an intrusion. In addition, derivation-constrained access control does not require a gateway to run access control, whereas the alternative approach does.

4.9 Related Work

There are several areas of research that are relevant to our work. In particular, there is related research about access control for delegation gateways, derivation of access rights, and computation on encrypted data.

We first review some related work dealing with protocol gateways. We have discussed the limitations of these schemes in Section 4.2. Sollins' solution [115] relies on "passports". In a passport, an entity states that some other entity can act on its behalf. A client generates a passport stating that the first gateway can act on her behalf. The client may also add constraints that limit the way the gateway can act to the passport. The gateway takes

the passport, adds any further constraints, signs it, and hands it off to the next gateway. This process is repeated until the passport arrives at the service, which verifies the passport. In Neuman's proxy-based authorization [97], entities hand out proxies (i.e., signed tokens) to other entities to grant them (potentially limited) access rights. Proxies can be used to implement Sollins' cascaded authentication. Gasser and McDermott [53] have a client issue a delegation certificate to a local workstation allowing the workstation to act on the client's behalf. The local workstation might issue another certificate to the next workstation. The delegation certificates expire in order to limit the period during which corrupt systems can operate on the client's behalf. There are also "delegation keys" that prevent a workstation from affecting clients upon a compromise taking place between the ending of a client session and the expiration of a delegation certificate. A workstation erases its received key when a session ends. Lampson et al. [81] also exploit delegation to grant access rights to gateways. They achieve delegation by modifying a service's access control list of the object to which access should be delegated. In addition, both a client and a gateway need to make a statement that lets the service conclude that the gateway is delegated an access right by the client. To cope with intruders, the authors suggest the use of expiring keys. Howell and Kotz [69] suggest the usage of "quoting protocol gateways" to prevent a gateway from making access decisions. Namely, principals do not delegate authority to the gateway, instead they delegate authority to the gateway quoting a particular client. Upon a request, the gateway needs to quote the identity of the client to the service, and the service uses this quoting information in its access control. Kornievskaia et al. [78] study the implementation of a protocol gateway that lets clients access Kerberized services using an SSL-enabled Web browser.

Bertino et al. [18] present a temporal authorization model, in which an access right becomes valid whenever an access right that it depends on becomes valid. In our scenario, the former access right could be a gateway's access right to information offered by a service and the latter one a client's access right to information provided by the gateway. However, the presented model applies only to a centralized environment, in which there is an administrator that manages both access rights and dependencies between access rights. In pervasive computing, individuals issue access rights and establish derivation relationships.

In derivation-constrained access control, an intruder into a gateway gets to see any information given to the gateway when a service answers an authorized request from the gateway. We could avoid this information leak by having the service encrypt information such that only the client can decrypt it and by having the gateway compute on the encrypted information. This approach will obviously work if the gateway does not need to perform any computations on the information (e.g., in the people-location scenario). The approach is also applicable to some very specific scenarios, where the gateway does have to run

some computations on the encrypted information. For example, Song et al. [116] discuss searching on encrypted data and Kissner and Song [77] show how to perform set operations on encrypted data. However, it is unclear how practical this approach is in general.

4.10 Summary

We introduced the concept of derivation-constrained access control, which allows one to limit the capabilities of an intruder into a gateway. We presented a formal model for expressing derivation-constrained access rights to information and derivation properties between information, and we proposed a mechanism for exploiting both of them in derivation-constrained access control. The sample implementation and its deployment demonstrate the feasibility of our design. The performance of access control is competitive. A large fraction of the cost is caused by expensive, but required operations for authenticating peers.

Derivation-constrained access control is applicable to gateways that query for input information only when receiving an authorized query for output information. However, the concept is not applicable to gateways that need to continuously ask for input information, even when there is no query for output information. For this class of gateways, an intruder will still be able to issue authorized queries for input information.

In the next chapter, we examine how we can support context-sensitive constraints on access rights, without leaking information if these constraints involve confidential information.

Chapter 5

Confidential Context-Sensitive Constraints

Access control ensures that only principals that have an access right to confidential information can access this information. Access rights can be constrained, that is, they are valid only if certain conditions are met. For example, an access right can state that access is granted only during office hours. Pervasive computing makes it possible to constrain access rights based on a person's context information. For example, an access right can declare that access should be granted to an individual only if the individual is at a particular location. However, context-sensitive constraints can lead to information leaks. In this chapter, we study information leaks that context-sensitive constraints can cause and how to avoid these leaks.

5.1 Overview

Let us look at three examples to illustrate information leaks caused by context-sensitive constraints. In our first example, information leaks to a service. Assume that Carol lets people standing in front of her office see her current calendar entry; a cellphone service provides people's location information, and a centralized calendar system offers Carol's calendar information. Given this setup, when Alice asks the calendar service for Carol's calendar entry, the calendar service could learn Alice's location while running access control, either by querying the location service directly or by being told that the constraint in Carol's access right is fulfilled. Therefore, Alice's location information could leak to the calendar service (i.e., to the organization running this service).

In the second example, information leaks to a person who asks for information. Assume that Carol allows people to access her calendar entry if she is in her office. Therefore, if somebody can retrieve this entry, he will also learn that Carol is in her office. A person planning on breaking into Carol's house would happily take advantage of this information leak.

In our third example, information leaks to a person who hands out access rights. Assume that Carol grants herself an access right to her calendar entry under the condition that Alice is at a particular location. When the calendar system grants Carol access to her entry, she will learn Alice's location, which could be an information leak.

Related work has largely ignored information leaks caused by context-sensitive constraints on access rights to information. In this chapter, we examine how constraints on access rights can cause information leaks and how to avoid these leaks. In particular, our contributions include

- a systematic investigation of different access-control approaches for information leaks caused by constraints,
- the organization of access rights and of constraints on them in a directed graph, which simplifies the detection of conflicting constraints and of information leaks and the resolution of constraints,
- the concept of hidden constraints, which makes it possible to keep constraints secret and thus reduces the danger of information leaks, and
- a formal model that supports constraints on access rights in distributed access control, a prototype implementation based on this model, and an evaluation of this implementation.

We start by listing the system model (Section 5.2) and by extending our formal model to support constrained access rights (Section 5.3). We then focus on a restricted set of constraints and discuss information leaks that these constraints could cause in different access-control approaches and how to avoid these leaks (Section 5.4). Based on this discussion, we drop the restrictions on constraints and introduce a graphical representation of access rights and of constraints on them and show how to exploit this graph for preventing information leaks (Section 5.5). We then examine how keeping constraints secret can also avoid information leaks (Section 5.6). Furthermore, we discuss interactions between information relationships and constraints (Section 5.7). Next, we present a constraint-aware, client-based access-control architecture (Section 5.8). Finally, we measure the performance of this architecture and discuss optimizations (Section 5.9).

5.2 System Model

In this section, we present the system model that we use for studying information leaks caused by context-sensitive constraints. In particular, we discuss the nature of constraints, the studied scenario, enforceability issues, and staleness of information used for constraining access.

5.2.1 Nature of Constraints

We assume that a constraint consists of information and of a set of permitted values for this information. The constraint is satisfied if the current value of its information equals one of the values in the set. A tuple of constraints attached to an access right is satisfied if each constraint in the tuple is satisfied. We observe that many sensible constraints in pervasive computing involve information about the context of a person. A person's context can include the current time, her current activity, or her current location. In addition, a constraint is typically either about the person that asks for information (e.g., "Carol grants Alice access to Carol's calendar if Alice is in Alice's office.") or about the person that grants an access right (e.g., "Carol grants Alice access to Carol's calendar if Carol is in Carol's office."), but not about third entities. Therefore, in this chapter, we are mainly interested in constraints that deal with context-sensitive information about the first two entities (though our presented solution is powerful enough to support constraints involving third entities). We focus on context-sensitive constraints that are confidential (e.g., a person's location or activity, but not the current time) and that have dynamic values, which makes it infeasible to check the satisfaction of a constraint upon the specification of an access right.

5.2.2 Scenario

In this chapter, we examine the following scenario: a client wants to retrieve information provided by a particular service, and this retrieval requires the retrieval of information from other services to ensure the satisfaction of constraints. We call the first type of information *primary information*; it is provided by the *primary service*. The second type of information is *constraint information*; it is provided by *constraint services*. Let us look at an example. Alice wants to access Carol's calendar entry. Her access right to this information is constrained to Alice being in her office. Carol's calendar and Alice's location information are offered by a calendar and a location service, respectively. Here, Carol's calendar entry corresponds to the primary information and Alice's location to constraint

information. The calendar service is the primary service and the location service is a constraint service. Note that for different requests, the same information can be either primary or constraint information, and a service can be either the primary or a constraint service.

Multiple services can provide the same type of information. For example, there are multiple ways to locate a person. For simplicity reasons, we assume that there is only one service that offers the primary information in our problem setting. Extending our algorithms to support multiple such services is straightforward. However, picking a constraint service requires some thought. We could let the owner of constraint information choose a service. However, a malicious owner can pick a service that always guarantees satisfaction of a constraint. In the calendar example above, Alice, as the owner of her location information, could designate a fake location service that always returns Alice's office as her location. Instead, it is the issuer (Carol in our example) of a constrained access right who should pick the constraint service(s) providing the constraint information in the access right.

5.2.3 Enforceability

To avoid an information leak, the client should be able to gather confidential knowledge about constraint information, such as its current value, only if the client had an access right to this constraint information. However, it is not always possible to enforce this requirement. For example, if the client had an access right to information constrained to particular values of this information (e.g., "Carol grants Alice access to Carol's location if Carol is in her office.") and if the client was denied access to the information, the client could infer a set of possible current values for the information, where this set is disjunct from the set of values that the client is permitted to learn. In the worst case, where the client is permitted to access all but one of the information's possible values, a denied request leaks the current value of the primary information to the client. We can generalize this case to the case where multiple access rights depend on each other. For instance, assume that there are two possible values for both Alice's location information and her activity information. Alice grants Bob access to her location information constrained to a particular value of her activity information. She also grants him access to her activity information constrained to a particular value of her location information. If Bob issues a query for Alice's location and is denied access, the probability that Alice is performing the activity not listed in Bob's access right to the location information will be twice as high as the probability that she is performing the listed activity. (It is possible for such a loop to involve more than two access rights.) In this chapter, we want to avoid information leaks for the enforceable cases, that is, cases where the information that an access right grants

access to is different from the constraint information in the access right and where access rights do not depend on each other.

5.2.4 Staleness of Constraint Information

To ensure satisfaction of constraints, the access-control algorithm could retrieve the current value of constraint information from a constraint service, validate that the corresponding constraint is satisfied, get the primary information from the primary service, and return this information to the client. However, if primary and constraint information are offered by different services, this algorithm will not guarantee that the constraint is still satisfied at the exact time when the primary information is accessed. Namely, the retrieved constraint information could have become stale in the meantime.

We can avoid this problem by choosing a timestamp and assigning this timestamp to all queries sent to the (time-synchronized) constraint and primary services. Here, a service no longer returns the current value of its offered information. Instead, it returns the value of the information at the time indicated by the timestamp. A timestamp can correspond either to a time in the past or to a time in the future.

For times in the past, a service now needs to keep state and remember past values of the information that it offers. For some services, this requirement is difficult to implement. For example, a location service could periodically poll the location of all the locatable individuals. However, the polled values do not necessarily let the service correctly deduce the location of an individual for a random timestamp in the past. We could restrict the granularity of the permitted timestamps and make a timestamp correspond to a time when polling occurs. However, if the polling frequency is low, clients will potentially be given old information. If the polling interval is high, the overhead caused by the (very often unnecessary) polling will also be high. Finally, being able to impose the same granularity on all services seems unrealistic.

Services do not need to keep state when timestamps correspond to a time in the future. In this approach, a notification step precedes the submission of a query to a service. In the notification step, the service is notified of the upcoming query. In particular, the notification lists a timestamp covering a time in the future. This time corresponds to the time for which the upcoming query will expect the value of the information provided by the service. (Obviously, this query should be sent only after the time indicated in the timestamp has passed.) In the example given above, after receiving a notification containing a timestamp, the location service polls the location of the queried user (and of no other users) at the indicated time. Later, when the actual query arrives, the service returns the polled value.

The problem with this approach is that interactivity could suffer. The timestamp submitted in the notification needs to correspond to a time in the future that is far away enough to guarantee that all the involved services can be contacted beforehand. Depending on the number of services, this could take hundreds of milliseconds and affect interactivity negatively. Furthermore, the approach makes communication more expensive. Each service needs to be contacted twice (first for submitting the notification, then for submitting the actual query). If we used the same connection for both transfers instead, we could tie up communication resources for a potentially long time.

As outlined above, avoiding staleness of constraint information raises several implementation challenges. We do not expect staleness of constraint information to become a severe issue for pervasive computing, as opposed to other environments (e.g., military or financial services). First, in pervasive computing, constraints involve context-sensitive information. Typically, this information changes little between the time that it is retrieved and the time that the information requested by a client is accessed. For example, since individuals move at a finite speed, the change in an individual's location between retrieving her location from a constraint service and accessing the primary service is limited. To prevent attackers from artificially prolonging this period, a service offering context-sensitive information should indicate for how long the service expects the information to keep its current value. The access-control algorithm should return primary information to the client only within the indicated time frame. Second, we can look at staleness as another factor that contributes to the uncertainty about the current value of information used in a constraint. In pervasive computing, there typically already is some uncertainty about context-sensitive information (e.g., location information can be wrong or conflicting), and access control needs to be able to deal with this uncertainty. We do not address the incorporation of uncertainty into access control in this thesis.

In our implementation, we do not use timestamps and have constraint services indicate for how long they expect constraint information to keep its current value.

5.3 Formal Model

In this section, we extend our formal modal to support constraints on access rights. Our extensions will make clear what statements an entity needs to issue and will serve as a guide for the design of a distributed access-control architecture that supports constraints (Section 5.8).

We have the following statement denote that principal A grants principal B access to information $C.x$ (assuming that A itself has access), given that constraint information $D.y$

has a current value in set $\mathcal{V}$, where A decides about the constraint service that provides $D.y$:

$$A \textbf{ says } ((A \textbf{ says } D.y \in \mathcal{V}) \supset (B \stackrel{C.x}{\Longrightarrow} A)). \tag{5.1}$$

It is possible for A to list multiple constraints. We explicitly include A as the principal that is responsible for guaranteeing satisfaction of $D.y \in \mathcal{V}$ since different issuers might want to use different constraint services for providing the same constraint information. Instead of listing its own identity as the principal responsible for guaranteeing satisfaction of the constraint, A could directly give the identity of the constraint service in the statement. However, we prefer to keep the mapping separate from the access right so that access rights do not have to be re-issued when the mapping changes.

For establishing a mapping, principal A states that a constraint service, S, can certify on its behalf that $D.y$ has a value in set $\mathcal{V}$, or

$$A \textbf{ says } S \stackrel{\{\text{“}D.y \in \mathcal{V}\text{”}\}}{\Longrightarrow} A.$$

For short,

$$A \textbf{ says } S \stackrel{D.y \in \mathcal{V}}{\Longrightarrow} A. \tag{5.2}$$

The mapping above depends on the set of values $\mathcal{V}$. Therefore, principal A would have to issue separate mappings for each set and each type of constraint information that A lists in an access right. However, we want to give A the option to define a single mapping for a particular type of constraint information, which can be used for all possible sets of values for this information. To achieve this property, we introduce an axiom stating that if S can certify membership in a set of values, S can also certify membership in subsets of this set, or

$$\vdash (E \textbf{ says } S \stackrel{D.y \in \mathcal{V}}{\Longrightarrow} A) \supset (E \textbf{ says } S \stackrel{D.y \in \mathcal{V}'}{\Longrightarrow} A) \text{ with } V' \subset V. \tag{5.3}$$

Principal A can now issue

$$A \textbf{ says } S \stackrel{D.y \in \mathcal{W}}{\Longrightarrow} A, \tag{5.4}$$

where $\mathcal{W}$ corresponds to the range of $D.y$, that is, $\mathcal{W}$ lists all possible values of $D.y$. Using Axiom (5.3), we can derive the mapping for any set $\mathcal{V}$ listed in an access right containing constraint information $D.y$ and issued by A.

Another useful axiom involves services that are not aware that they are used for guaranteeing satisfaction of a constraint. For example, service S might just return the current value of $D.y$ (e.g., $D.y = v$) instead of acknowledging that this current value is in a particular set (e.g., $D.y \in \mathcal{V}$). To be able to exploit such a service, we introduce an axiom that states

$$\vdash (S \textbf{ says } D.y = v) \supset (S \textbf{ says } D.y \in \mathcal{V}) \text{ with } v \in \mathcal{V}. \tag{5.5}$$

Let us now illustrate how access control uses the additional axioms by walking through an example scenario in the context of client-based access control.

Principal A defines the access right and the mapping from constraint information to a constraint service given in Statements (5.1) and (5.4), respectively. Client B wants to access information $C.x$. The principal contacts constraint service S and retrieves the current value of constraint information $D.y$, that is,

$$S \textbf{ says } D.y = v. \tag{5.6}$$

For example, S issues a digital certificate expressing this statement.

Principal B then exploits this statement in its proof of access for information $C.x$. In particular, with the help of Axiom (5.5), we can derive

$$S \textbf{ says } D.y \in \mathcal{V}. \tag{5.7}$$

Furthermore, using Axiom (5.3) and the mapping of constraint information $D.y$ to constraint service S in Statement (5.4), we conclude that this mapping also applies to the set of values given in access right (5.1), that is, we derive Statement (5.2).

We can derive a speaks-for relationship regarding the statement $D.y \in \mathcal{V}$ from Statement (5.2). Based on this relationship and Statement (5.7), we conclude

$$A \textbf{ says } D.y \in \mathcal{V}. \tag{5.8}$$

Using the constrained access right (Statement (5.1)) and Statement (5.8) guaranteeing satisfaction of the constraint, we can use modus ponens for **says**[1] to derive

$$A \textbf{ says } B \stackrel{C.x}{\Longrightarrow} A,$$

from which we can infer speaks-for relationship

$$B \stackrel{C.x}{\Longrightarrow} A.$$

Principal B can use this relationship in its proof of access, as explained in Sections 2.5.2 and 2.6.2.

For access rights where the primary information is identical to constraint information, the client will not be able to resolve the corresponding constraint before sending its proof of access to the primary service. Here, the client will insert a dummy statement showing constraint satisfaction into the proof. The primary service will replace this dummy statement with a valid statement, if possible.

5.4 Constraints and Information Leaks

Information leaks due to constrained access rights occur when a single entity or multiple, collaborating entities that are familiar with a constraint specification in an access right and that are able to observe requests exploiting this access right can infer confidential knowledge about the constraint information listed in the specification that the single entity and all of the collaborating entities, respectively, are not authorized to access. (If any of the collaborating entities is authorized, this knowledge does not result in an information leak, since the authorized entity could always proxy for the unauthorized entities.) In particular,

[1] $\vdash (A \textbf{ says } s \wedge A \textbf{ says } (s \supset s')) \supset (A \textbf{ says } s')$ [81].

the leaked knowledge reveals that the current value of the constraint information is in the set of values listed in the constraint specification or that the current value is not in this set. In inference-control research [46], these two cases are called *positive* and *negative compromises* [32], respectively. We assume that the range of values that constraint information can have is publicly known. Therefore, both for positive and negative compromises, a set of possible current values leaks. If this set contains only one element, the current value leaks, and there is an *exact compromise* [3]. If this set contains more than one element, there is still a *partial compromise* [3], since the set is smaller than the range of values that the constraint information can have.

There are multiple approaches to deploy access control in a distributed environment. In this section, we investigate how information leaks, as defined above, can occur in three different approaches, and we present access-control algorithms to avoid these leaks.

We concentrate on the simple case where the client's access right to the primary information has a set of constraints attached to it (e.g., "Carol grants Alice access to her calendar entry if Carol is in her office.") and where the client's access rights to constraint information are unconstrained (e.g., "Carol grants Alice unconstrained access to her location information."). We discuss the more general case in Section 5.5.

In this section, whenever we state that an entity has an access right to information, we have this statement cover both the case where access is granted based only on an initial access right and the case where access is granted based on an initial access right and some forwarded access rights.

5.4.1 Centralized Access Control

In centralized access control, the client sends its request for the primary information to a centralized entity, which runs access control on behalf of individual services. If there is an access right that grants access to the client, the centralized entity will ensure that all the constraints in the access right are satisfied. (We assume that access rights are stored with the centralized entity.) Therefore, the centralized entity has to retrieve the current values of the constraint information from the corresponding constraint services. If all the constraints are satisfied, the centralized entity will retrieve the primary information from the primary service and return the information to the client.

For information leaks, as defined above, to occur, an entity needs to know the constraint specifications in an access right. In centralized access control, we assume that, in practice, the client being granted a (constrained) access right has this knowledge. In addition, the issuer of the access right obviously also has this knowledge. (So does the centralized

entity, but it has access to any information, and information cannot leak to it.) Therefore, we have to prevent confidential knowledge about constraint information from leaking to the client and to the issuer of an access right.

The client could derive this confidential knowledge by observing whether its request for the primary information is granted or denied access. In the first case, the client will learn that all the constraints in the client's access right are satisfied. Therefore, for each constraint, the client will be able to infer at least a set of possible current values. In the second case, the client will learn at least that one of the constraints in the client's access right is not satisfied and at most the current value of the (single) constraint in the access right. In both cases, there could be an information leak. To avoid this leak, the centralized entity needs to ensure that the client has access to the constraint information listed in the access right. In more detail, the centralized entity implements the following steps:

1. Deny access if there is no access right that grants the client access to the primary information.

2. For each constraint in this access right,

   (a) deny access if there is no access right that grants the client access to the constraint information listed in the constraint, and

   (b) retrieve the current value of the constraint information from the corresponding constraint service and deny access if this value is not in the set of permitted values listed in the constraint specification.

3. Retrieve the current value of the primary information from the primary service. Return the value to the client if the client's access right does not restrict the permitted values or if one of the permitted values equals the current value, else deny access.

In this algorithm, step 2 will deny access to the client if the client does not have access to constraint information, and the client cannot infer any confidential knowledge about the constraint information from the denial.

Let us now discuss how an issuer of an access right can gather confidential knowledge about constraint information in the access right. The issuer of the client's access right to the primary information can collude with the primary service (i.e., its administrator). Whenever this service is contacted by the centralized entity, the constraints in the client's access right must be satisfied, and the issuer and the primary service could infer confidential knowledge about constraint information. Therefore, based on our definition of an information leak, it seems sufficient to have the centralized entity ensure that either the

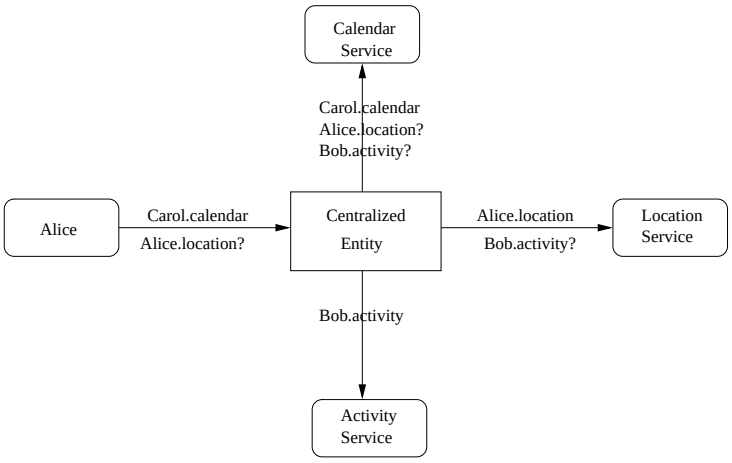


Figure 5.1: Information leak to issuer of access right. The scenario illustrates Bob's activity information leaking to Alice. Arrows indicate queries for information. The first line associated with an arrow gives the queried information. The second (and third) line lists the information that the queried service (or Alice) needs to have access to. This requirement needs to be verified by the centralized entity before the entity sends a query to a service (or answers Alice's request.)

issuer or the primary service has access to the constraint information in the client's access right before retrieving the primary information from the primary service.

However, the second option does not entirely prevent confidential knowledge about constraint information from leaking to the issuer of an access right. Figure 5.1 illustrates such a scenario. Assume that Carol grants Alice access to her calendar if Alice is in Alice's office. As required by the second option, upon Alice issuing a request, the centralized entity ensures that the calendar service has access to Alice's location before retrieving Carol's calendar entry from the calendar service. This service has an access right to Alice's location information issued by Alice constrained to Bob being not busy. Therefore, the centralized entity must also ensure that the calendar service has access to Bob's activity information. Furthermore, before retrieving Alice's location from the location service, the centralized entity must ensure that the location service can access Bob's activity information. If Bob grants the calendar and location services access to his activity information, Alice's request will eventually be granted access. Then, Alice can conclude that Bob is not busy, regardless of whether she has an access right to this information. To avoid this kind of information leaks, the centralized entity must ensure that the issuer of an access right has access to constraint information in the access right. This condition will prevent

Alice from exploiting the access right to the calendar service constrained based on Bob's activity for learning this activity information.

In order to validate that the issuer of an access right has access to constraint information, the centralized entity employs the same algorithm as the one it uses for ensuring that the client has access to constraint information given before. Note that the centralized entity must recursively ensure that the issuer of an access right to the issuer of the client's access right has access to the constraint information in that access right.

The information leak just described occurs because Alice knows the constraint specification in the calendar service's access right. In the given scenario, Alice has this knowledge because she issued this access right. If the contents of access rights were public, Alice would have this knowledge even if the access right had been issued by someone else. In this case, ensuring that the issuer of an access right has access to constraint information in the access right would not prevent confidential knowledge about this information from leaking to Alice. This observation suggests that access rights should not be public and that they are kept confidential. In general, we do not expect access rights to be public, since issuing an access right is often a personal decision, which should be kept private.

Similarly, if access rights were public, the primary service could learn confidential knowledge about constraint information in the client's access right to the primary information without having to collude with the issuer of the access right.

In summary, the centralized entity must validate that the client has access to the constraint information in the client's access right to the primary information and that the issuer of an access right has access to the constraint information in the access right. Furthermore, access rights should be kept confidential.

5.4.2 Service-Based Access Control

In service-based access control, the client asks the primary service directly for the primary information. The primary service has to ensure that there is an access right for the client. For a constrained access right, the service has to contact the corresponding constraint services and check satisfaction of the constraints. The constraint services also run access control.

In service-based access control, we assume that, in practice, the client being granted a (constrained) access right knows the contents (e.g., constraint specifications) of this access right. In addition, a service knows the contents of access rights to information that the service provides and to information that the service is authorized to access. Finally, an issuer of an access right obviously also knows the contents of the access right. Therefore,

we have to prevent confidential knowledge about constraint information from leaking to the client, to services, and to the issuer of an access right.

We can use the algorithm introduced in Section 5.4.1 to prevent the client from deriving confidential knowledge about constraint information. Here, it is the primary service that runs the algorithm. While resolving constraints in the client's access right, the primary service contacts corresponding constraint services and becomes a client itself. The constraint services ensure that the primary service has access to the requested constraint information, so this information cannot leak to the primary service. Furthermore, the constraint services must avoid that confidential knowledge about constraint information in the primary service's access rights leaks to the primary service, using the algorithm given above. While resolving constraints in these access rights, the constraint services recursively might have to contact additional constraint services.

Similar to centralized access control, ensuring that a service has access to constraint information in an access right to the information offered by the service, as service-based access control does by design, is not sufficient to prevent confidential knowledge about constraint information from leaking to the issuer of an access right. For example, the scenario discussed in Section 5.4.1 is also relevant for service-based access control. Therefore, before answering a request for information, a service must ensure that the issuer of an access right to the information has access to the constraint information listed in the access right.

Furthermore, there could be additional information leaks if access rights were public. If the client knew the contents of the primary service's access rights to the constraint information listed in the client's access right to the primary information (and, recursively, the contents of other service's access rights), confidential knowledge about constraint information in a service's access right could leak to the client. Again, not making access rights public avoids this problem.

A service must ensure that the issuer of an access right has access to constraint information listed in the access right. Therefore, the service must know some of the issuer's access rights. However, it can be difficult for the service to learn about these access rights since, as mentioned above, access rights should not be public. This observation suggests that the types of constraints listed in an access right should be kept simple. Namely, we expect most confidential context-sensitive constraints to involve either the subject or the issuer of an access right, but not other entities. In the first case, the issuer by default has access to the constraint information. In the second case, the client (or a service) can inform a queried service of its access rights.

In summary, a service must ensure that the client (or another service) has access to

the constraint information in the client's (or service's) access right to the requested information and that the issuer of an access right has access to the constraint information in the access right. Furthermore, access rights should be kept confidential and the types of constraints simple.

5.4.3 Client-Based Access Control

In client-based access control, the client needs to prove to the primary service that it is authorized to access the primary information. The service makes the access decision by validating this proof of access. The proof lists the client's access right to the primary information and confirms that each of the constraints in the access right is satisfied. Therefore, before contacting the primary service, the client first has to build proofs of access for constraint information, contact the corresponding constraint services, and have a service confirm that a constraint is satisfied. We use the term *assurance* for representing such a confirmation. (We present ways to implement such assurances in Section 5.8.)

In client-based access control, a client being granted a (constrained) access right knows the contents (e.g., constraint specifications) of this access right. A service can learn the contents of access rights listed in a proof of access. Finally, an issuer of an access right obviously also knows the contents of the access right. Therefore, we have to prevent confidential knowledge about constraint information from leaking to the client, to services, and to the issuer of an access right.

While resolving the constraints in its access right to the primary information, the client needs to build proofs of access for the corresponding constraint information. Therefore, confidential knowledge about this constraint information cannot leak to the client.

The primary service can learn the constraint specifications in the client's access right to the primary information by looking at the proof of access. Since the proof also contains an assurance for each constraint, confidential knowledge about the constraint information could leak to the primary service. Therefore, the client must validate that the primary service has access to the constraint information before sending the proof to the service.

For this validation, the client must know the primary service's access rights to the constraint information. For each access right, the client has to validate its constraints, which requires the client to retrieve constraint information from a constraint service, using access rights issued to the client. If the issuer of such an access right colluded with the primary service, they would know that whenever the primary service is contacted, the constraints in this access right are satisfied and they could derive confidential knowledge about constraint information in the access right. Therefore, the client must also ensure that

the issuer of an access right has access to constraint information in the access right.

If access rights were public, the primary service could exploit the information leak just described without having to collude with the issuer of an access right. Again, this observation suggests that access rights are not made public. However, building a proof of access requires the client to know the primary service's access rights to the constraint information listed in the client's access right to the primary information. We can solve this conflict in the same way as for the service-based approach, that is, by keeping the types of constraints listed in an access right simple and by having constraints involve either the subject or the issuer of an access right. In the first case, the client can decide whether it wants to grant access to the primary service. In the second case, when issuing an access right, the issuer could also inform the subject of the services that it grants access to its constraint information listed in the access right. Another option are hidden constraints, which prevent a service from learning the constraint specification in the first place (see Section 5.6).

In summary, the client must ensure that the primary service has access to the constraint information in the client's access right to the primary information and that the issuer of an access right has access to the constraint information in the access right. Furthermore, access rights should be kept confidential and the types of constraints simple.

5.4.4 Third Entity

For client-based and service-based access control, it is possible for the client and a service, respectively, to have a separate entity deal with constraints on their behalf. This entity retrieves constraint information from constraint services and lets a client or service know whether a constraint is satisfied.

This case raises similar issues as the other scenarios. If the separate entity had the same set of access rights as the client or the service on whose behalf it acts, this case would be identical to having the client or service resolve constraints directly. If the separate entity had more access rights than a client or a service, we would risk information leaks similar to the case of centralized or service-based access control, where the third entity is now in the role of the centralized entity or the service.

Apart from the presented access-control approaches, hybrid approaches are possible. For example, in client-based access control, the primary service could exploit client-based access control, whereas a constraint service adopts service-based access control. Our presented algorithms also apply to hybrid approaches. In the example, the constraint service ensures that no constraint information leaks to the client and the client validates that no

constraint information leaks to the primary service.

5.5 Access-Rights Graphs

We have assumed that the client's access rights to constraint information are unconstrained. However, in the more general case, such an access right can be constrained, as discussed in this section.

Our discussion in the previous section revealed that making access rights public can result in information leaks that are difficult to avoid. Therefore, in the rest of this chapter, we assume that access rights are not public. Namely, only the issuer of an access right and the subject being granted access (e.g., the client) know about the access right. For centralized and service-based access control, we assume that the centralized entity and a service offering the information, respectively, are also aware of the access right.

5.5.1 Design

Our access-control algorithms for the general case exploit *access-rights graphs*. An access-rights graph captures relationships between access rights and constraints on them. The graph allows for easy detection of potential problems, such as information leaks, loops, or conflicting constraints, and simplifies resolution of constraints.

An access-rights graph is built for particular information in terms of an entity's access rights. The graph represents the conditions under which this entity should be granted access to the information. The edges and nodes of the graph are derived from the entity's access rights. In particular, a node in the graph represents information, and the edges outgoing from a node denote the constraints on an access right (or on a chain of access rights) to the information in the node. An edge has a set of values attached to it, meaning that the information in the node that the edge is pointing to is constrained to the values in the set. If an access right to information is unconstrained, the corresponding node has an outgoing edge that goes back to the node and that is marked with "$*$"; such a node cannot have more than one outgoing edge. We call the node containing the information for which the graph is built *root node*. Figure 5.2 shows an example of an access-rights graph. For simplicity reasons, we assume that all issuers of access rights agree on the service that provides a particular type of constraint information. Note that since the access right to $C.x$ is constrained to particular values of this information, we cannot necessarily avoid confidential knowledge about $C.x$ from leaking in case of a denied request, as outlined in

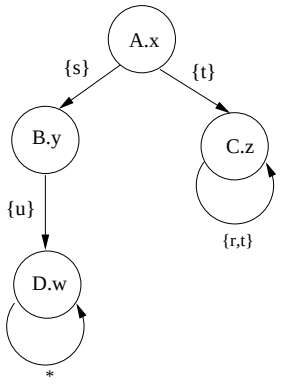


Figure 5.2: Access-rights graph. The graph is for information $A.x$ in terms of E's (shortened) access rights A **says** $((B.y \in \{s\} \wedge C.z \in \{t\}) \supset (E \stackrel{A.x}{\Longrightarrow} A))$, B **says** $((D.w \in \{u\}) \supset (E \stackrel{B.y}{\Longrightarrow} B))$, C **says** $((C.z \in \{r,t\}) \supset (E \stackrel{C.z}{\Longrightarrow} C))$, and D **says** $E \stackrel{D.w}{\Longrightarrow} D$. The node listing $A.x$ is the root node.

Section 5.2.3.

We call an access-rights graph *conflict-free* if for nodes with multiple incoming edges, the intersection of the sets of values attached to these edges is not empty. Figure 5.4 shows an example of a graph with a conflict. There could be multiple graphs for the same information in terms of an entity's access rights if the entity had multiple access rights to this information, but with different constraints on them.

The entity whose access rights are used for building an access-rights graph should be granted access to the information in the root node if 1) each node has at least one outgoing edge (i.e., there is an access right to the information in the node), 2) the graph is conflict-free, and 3) the current value of the information in each node is listed in each of the node's incoming edges.

Assuming that an entity's access rights are locally available, building a conflict-free access-rights graph out of these access rights is a completely local step. We present the pseudocode in Figure 5.3. Ensuring that each constraint is satisfied requires traversal of the graph and contacting remote services that offer the information in a node. We call this graph traversal *resolution*.

Access-rights graphs can become arbitrarily complex. At present, we expect them to be rather simple for practical scenarios, such as the graph shown in Figures 5.2. This expectation is based on two observations: First, people typically need to specify access

```
// The algorithm assumes that nodes contains the current set of nodes in the access-
// rights graph. To start with, nodes contains the root node of the graph and the
// algorithm is called for this node. In addition, rights contains the access rights of the
// entity in terms of which the graph is built. The algorithm also assumes the existence
// of a routine boolean has_conflict(Node node, Values values) that
// returns true if the intersection of the values attached to any incoming edge of node
// and values is empty.

// Return true if there is conflict-free access-rights graph for the information in the node.
boolean build_graph(Node current_node)
{
  // Retrieve access rights to information in current_node.
  // (There could be multiple access rights (and thereby graphs).)
  Set rights = rights.retrieve(current_node.get_information());
  while (rights.notEmpty()) {
    AccessRight right = rights.remove();
    Constraints constraints = right.get_constraints();
    while (constraints.notEmpty()) {
      Constraint constraint = constraint.remove();
      // If there already is a node for the information in the constraint, retrieve it.
      Node node = nodes.retrieve(constraint.get_information);
      if (node != null) {
        // Go through node's incoming edges and check for conflict with values
        // permitted by constraint.
        if (has_conflict(node, constraint.get_values()) break ;
        // Establish an edge from current_node to existing node listing values in constraint.
        nodes.establish_edge(current_node, node, constraint.get_values());
      } else {
        // Establish new node for constraint information and edge to it from current_node.
        Node new_node = nodes.establish_node_and_edge(constraint,
          current_node);
        // Try to build subgraph for information in this node.
        if (!(build_graph(new_node))) {
          nodes.remove_node_and_edge(new_node);
          break ;
        }
      }
    }
    return true;
  }
  return false;
}
```


Figure 5.3: Building of access-rights graphs. The pseudocode gives the algorithm for building a conflict-free access-rights graph.

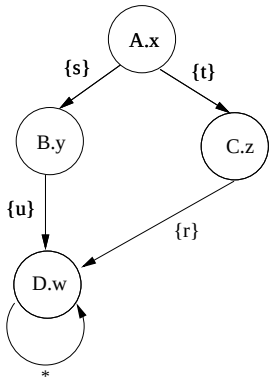


Figure 5.4: Access-rights graphs with conflict. The graph has conflicting constraints on $D.w$.

rights in a manual way, which tends to lead to simple access rights, having no or only a few, broad constraints. Second, the amount of context information that is currently available about people and that can be used to constrain access is still rather limited. However, both observations probably will no longer hold in the future. For example, if users let electronic agents manage access rights on their behalf, access rights will become more complicated and can involve more and narrower constraints. Also, the amount of information available about people is steadily increasing.

Access-rights graphs are required regardless of the access-control approach. Let us now discuss how we employ these graphs in centralized, service-based, and client-based access control.

5.5.2 Centralized Access Control

As outlined in Section 5.4.1, centralized access control must ensure that the client can access primary and constraint information and that the issuer of an access right can access constraint information in the access right. For both validations, the centralized entity builds and resolves access-rights graphs. First, there has to be a graph for the primary information in terms of the client's access rights. Second, when resolving this graph, the entity must ensure that, for each node in this graph, the issuer(s) of the (chain of) access right(s) to the information in the node has access to the information in the nodes pointed to by that node, which requires building and resolving additional graphs for each node. Similarly, resolution of these graphs might recursively require additional graphs. We give

the pseudocode in Figure 5.5. Resolution of a graph starts at a node with no outgoing edges other than an edge pointing to itself and works its way toward the root node. If the graph had a loop involving more than one node, resolution could start at any node in the loop (not covered by the pseudocode).

For example, for the graph given in Figure 5.2, the centralized entity must ensure $D.w = u$ and that B has access to this information, which requires building and resolving an access-rights graph for $D.w$ in terms of B's access rights. Similarly, the entity has to check whether $B.y = s$ and $C.z = t$ and whether A has access to this information, that is, it must build build and resolve graphs for $B.y$ and $C.z$ in terms of A's access rights.

An access-rights graph is built for information in terms of an entity's access rights. Since access control might trigger recursive building of graphs, it is possible that the same graph is built multiple times and that loops occur. For example, assume that Bob grants Alice access to his location under the constraint that she is at a particular location and that Alice grants Bob access to her location under the constraint that he is at a particular location. If Alice wants to access Bob's location, the centralized entity will build and resolve a graph for Bob's location in terms of Alice's access right. To prevent constraint information from leaking to Bob, the resolution will trigger building and resolution of a graph for Alice's location in terms of Bob's access right, which will trigger building of the first graph again, and so on. The centralized entity can detect such a loop and resolve it.

5.5.3 Service-Based Access Control

In service-based access control, a service must ensure that the client (or another service) has access to the requested information and to the constraint information in the access right to this information. Furthermore, a service must ensure that the issuer of an access right has access to constraint information in the access right

Therefore, the primary service builds a conflict-free access-rights graph for the primary information in terms of the client's access rights. It then resolves the graph and retrieves constraint information from constraint services. These services will build and resolve access-rights graphs in terms of the primary service's access rights, which could require contacting additional constraint services. Furthermore, a service needs to ensure that the issuer of an access right has access to constraint information in the access right. A service running access control exploits the algorithm shown in Figure 5.5. Since access rights are not public, the client (or a service) must inform a contacted service of its access rights in order for the service to be able to build a graph.

```
// Return true if principal has access to information at given value.
boolean can_access(Principal principal, Information information,
 Value value)
{
 Set graphs = conflict-free access-rights graphs with at least one outgoing edge
   per node for information in terms of principals's access rights. If
   value != null and information in root node of a graph is constrained to particular
   values of this information, value must be contained in these values.
 while (graphs.notEmpty()) {
  Graph graph = graphs.remove();
  if (is_resolvable(graph)) return true;
 }
 return false;
}

// Return true if the constraints in the access-rights graph are satisfied and if there
// are no information leaks.
boolean is_resolvable(Graph graph)
{
 // Gather nodes that can be resolved.
 Set readySet = all nodes in graph with no outgoing edges other than an edge
   to itself;
 while (readySet.notEmpty()) {
  Node node = readySet.remove();
  Information information = node.get_information();
  Value value = retrieve current value of information from constraint service;
  if (value is not listed in all incoming edges of node) return false;

  // Ensure that issuers can access constraint information. (For client-based access
  // control, also ensure that service can access this information.)
  parents = nodes with an outgoing edge to node;
  while (parents.notEmpty()) {
   Node parent = parents.remove();
   Set issuers = issuers of access rights to information in parent;
   while (issuers.notEmpty()) {
    Principal issuer = issuers.remove();
    if (!(can_access(issuer, information, value))) return false;
   }
   if (all nodes with incoming edge from parent have been removed from readySet)
    readySet.add(parent);
  }
 }
 return true;
}
```


Figure 5.5: Access-control algorithm. Access control consist of building a conflict-free access-rights graph and of resolving this graph. In addition, access control must recursively ensure that issuers of access rights can access constraint information.

5.5.4 Client-Based Access Control

In client-based access control, the client must build proofs of access for the primary and constraint information. In particular, the client builds a conflict-free access-rights graph for the primary information in terms of its access rights and assembles proofs for the various nodes while resolving this graph. In addition, the client must ensure that no confidential knowledge about constraint information leaks to a service receiving a proof of access or to an issuer of an access right. Therefore, for each node in the graph, the client builds additional graphs for the information in the nodes pointed to by that node in terms of the access rights of the service offering the information in that node. Similarly, for each node, the client builds additional graphs for the information in the nodes pointed to by that node in terms of the access rights of the issuer(s) of the (chain of) access right(s) associated with that node.

A client implements the algorithm shown in Figure 5.5. There are two differences in `is_resolvable()` when compared with the centralized approach: First, as mentioned above, the client must check that a service can access constraint information, in addition to the issuer of an access right. Second, a client needs to build proofs of access while resolving an access-rights graph. In particular, when contacting a constraint service, the client will receive an assurance stating that a constraint is satisfied. Once it has received assurances for all the nodes that a node is pointing to, it can build a proof of access for the information in this node and contact the corresponding constraint (or primary) service.

For example, in the graph shown in Figure 5.2, the client first retrieves an assurance for $D.w = u$ from the constraint service offering $D.w$, using its access right as a proof of access. The client then uses this assurance and its access right to $B.y$ to build a proof of access for getting an assurance for $B.y = s$. Similarly, it gets an assurance for $C.x = t$. These two assurances and the access right to $A.x$ allow the client to build a proof of access for $A.x$. The service offering $A.x$ validates the proof and returns the current value of $A.x$.

Proof building becomes difficult for conflict-free access-rights graphs with loops involving more than one node, since there is no obvious node at which a client can start resolution. There are multiple ways to deal with such cases. If the information of all the nodes in the loop was offered by the same service, a client could have this service resolve the loop. If multiple services offered this information, a client could contact some of these services and ask them to resolve the constraints on its behalf. This option requires trust relationships between the services so that they can exchange constraint information. None of this constraint information must leak to the client unless all the constraints are satisfied.

5.6 Hidden Constraints

We have assumed that the issuer of an access right, the subject of the access right, the centralized entity in centralized access control, and a service offering the information that the access right grants access to in service-based access control know about the contents, such as constraint specifications, of the access right. Also, in client-based access control, a service will learn the contents of an access right that is part of a proof of access.

If an entity knows the constraint specifications in an access right, we must prevent constraint information from leaking to this entity. We can avoid these checks by keeping constraint specifications secret from the entity, that is, by employing *hidden constraints*. A hidden constraint keeps (parts of) its constraint specification secret from an entity. If a constraint specification is hidden from the entity, the entity cannot infer confidential knowledge about the constraint information listed in the specification, even if the entity is able to observe requests exploiting an access right containing the hidden constraint. Note that hidden constraints do not hide the existence of a constraint in an access rights from an entity, they hide only its specification.

Hidden constraints prevent us from having to ensure that the entity from which the constraint specification is hidden has access to the constraint information. An additional benefit is that hidden constraints can involve information to which the entity does not have access in the first place. For example, in client-based access control, assume that Alice uses a trusted service to provide important information about her. Alice grants Bob an access right to this information, given that he is at a particular location. If the constraint was not hidden from the service and Bob did not trust the service, Bob would be in a dilemma: Either he has to release his location information to the untrusted service or he cannot learn Alice's important information. Hiding the constraint from the service avoids this dilemma.

Hiding a constraint specification from an entity does not mean that the entity cannot finally learn the specification. If the entity had access to the constraint information in the specification, it could learn the specification by observing the system. However, this is not an information leak, since the entity has access to the constraint information.

5.6.1 Design

Let us now explore which parts of a constraint specification we can hide from which entity for the three discussed access-control approaches.

Obviously, we cannot keep the constraint specifications in an access right secret from

the issuer of the access right or from the centralized entity in centralized access control. However, it is possible to hide (parts of) the constraint specifications from other entities. A constraint specification consists of constraint information, a set of permitted values, and the identity of the constraint service responsible for acknowledging constraint satisfaction. Ideally, we keep the entire specification secret from an entity. However, in some cases, it is possible only to keep the first two items secret.

In centralized access control, we can hide a constraint specification in an access right entirely from the subject of the access right by keeping the access right secret from the subject.

In service-based access control, we can hide a constraint specification entirely from the subject of an access right in the same way as in centralized access control. It is not possible to hide the specification entirely from a service since the service must know the identity of the constraint service responsible for resolving the constraint. We can hide only the constraint information and the set of permitted values from the service. (For example, the issuer of an access right encrypts the two items with the public key of the responsible constraint service.) However, depending on the type of constraint information or service, knowing the constraint service might allow a service to deduce the type of constraint information (e.g., when a constraint service provides only one type of information), the owner of the constraint information (e.g., when a constraint service provides only one individual's information), or even the value of the constraint information (e.g., when a constraint service has limited coverage, such as a location service covering only one building).

In client-based access control, we can exploit the same reasoning as for a service in service-based access control to hide constraint information and a set of permitted values from the client. Again, since we cannot hide a constraint specification entirely from the client, it might still be possible for the client to deduce parts of the hidden specification. In addition, we can hide a constraint specification entirely from a service. Namely, a service is not interested in this specification; it wants to know only whether a constraint is satisfied. To support this feature, the issuer of an access right needs to associate a constraint with the access right such that a service cannot learn the constraint specification when looking at the access right in the proof of access, but the client building this proof remains able to gather assurances for the constraint. Similarly, a service must not be able to learn the constraint specification when looking at such an assurance. We present a possible implementation in Section 5.8.2. Such a hidden constraint prevents confidential knowledge about constraint information in an access right from leaking to a service. Note that the client still needs to ensure that this knowledge does not leak to the issuer of the access right.

5.6.2 Formal Model

When hiding constraints from an entity, we can use the same formal model as the one introduced for non-hidden constraints, but we have to change the representation of constraints. Clearly, we cannot use $B.y \in \mathcal{V}$ to express a constraint involving $B.y$.

Let us discuss the formal model for hiding constraint specifications from a service in client-based access control. We require that a service must not learn the identity of a service that acknowledges the satisfaction of a constraint, since this knowledge might allow inferring the type of constraint. For example, a constraint service cannot use its private key for signing an assurance.

We use $\tilde{H}$ for expressing hidden constraints. $\tilde{H}$ is a reference to a constraint specification, but must not leak any information about the specification and the constraint service responsible for the resolution of the constraint. In addition, we require that for each $\tilde{H}$, there is a $\tilde{H}^{-1}$ that can be used for expressing satisfaction of a hidden constraint. With the help of $\tilde{H}$, such a statement can be validated. For example, $\tilde{H}$ can be a public key and $\tilde{H}^{-1}$ the corresponding private key. We present two approaches, one based on digital certificates and another one based on one-way chains, in Section 5.8.2. Statement (5.1) expressing an access right can be rewritten using a hidden constraint as follows:

$$A \textbf{ says } ((A \textbf{ says } \tilde{H}) \supset (B \stackrel{C.x}{\Longrightarrow} A)). \tag{5.9}$$

Based on our requirements, A implicitly lets $\tilde{H}$ guarantee satisfaction of the constraint specification referenced by $\tilde{H}$, or

$$A \textbf{ says } \tilde{H} \stackrel{\tilde{H}}{\Longrightarrow} A.$$

5.6.3 Discussion

We can now summarize what information leaks we have to avoid for the different access-control approaches. We present this summary in Table 5.1. For an entity running access control (or building a proof of access in the case of client-based access control) depending on a constrained access right, this entity needs to ensure that the principals listed in the table have access to the different types of constraint information in the access right.

Approach	Constraints	
	Non-hidden	Hidden
Centralized	Client, issuer	Issuer
Service-based	Client, issuer	Issuer
Client-based	Service, issuer	Issuer

Table 5.1: Information leaks. For each approach, we show the potential information leaks due to deduction of constraint information. For centralized and service-based access control, constraints are hidden from a client. For client-based access control, they are hidden from a service.

5.7 Constraints and Information Relationships

Apart from constrained access rights, we could also have constrained information relationships, that is, an information relationship is valid only if certain conditions are met. However, the usefulness of this feature seems limited. In terms of bundling-based relationships, we strive to keep these relationships simple, since they will be defined by individuals who are not necessarily experts in access control. In terms of combination-based relationships, constraints on these relationships do not make sense since a complex piece of information either consists of certain other information or it does not; there are no conditions on this relationship. Therefore, we refrain from incorporating constraints into the establishment of information relationships.

5.8 Architecture

We now present a client-based access-control architecture that supports access rights with context-sensitive constraints. Our architecture supports hiding constraints from a service, not from a client. We give an overview of our architecture, take a closer look at the implementation of hidden constraints, and discuss how we use digital certificates for expressing various statements.

5.8.1 Overview

A client builds a proof of access. Figure 5.6 gives an overview of the client's components involved in proof building. The access-rights graph component is responsible for building

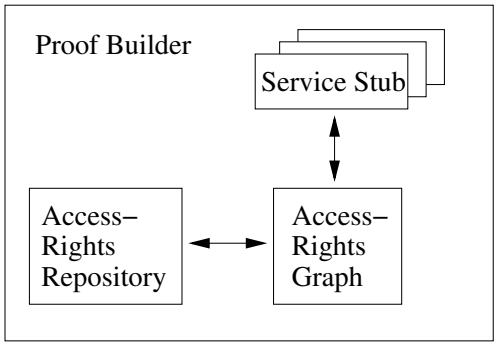


Figure 5.6: Proof-building architecture. The access-rights graph component interacts with the access-rights repository for building a graph and with service stubs for resolution of the graph.

and resolving access-rights graphs. This component implements the algorithm given in Figure 5.5. In order to build a graph, the component interacts with the access-rights repository. A service stub knows how to interact with a particular service. While resolving a graph, the access-rights–graph component asks service stubs to get either an assurance (for nodes other than the root node) or the value of the information (for the root node).

5.8.2 Hidden Constraints

We now discuss how we hide constraints from services. The issuer of a constrained access right includes only a reference to the constraint specification, but not the actual specification, in the access right. The issuer gives both the access right and the constraint specification to the client. The client resolves the constraint by presenting the specification to the corresponding constraint service, which will issue an assurance containing a reference to the constraint specification. The client includes both the access right and the assurance in its proof of access. The primary service must ensure that the access right and the assurance are valid and that they refer to the same constraint specification.

There are multiple ways to implement such a scheme. In our architecture, we rely on either digital certificates or one-way chains. We now explain the two options.

Digital Certificates

In the first approach, a constraint service signs a statement declaring that a constraint is satisfied. The signature has a lifetime corresponding to the time frame during which the constraint service believes the constraint to remain satisfied, as explained in Section 5.2.4. This approach has the advantage that a service can freely choose the lifetime of its signatures, depending on the service's domain-specific knowledge. The drawback of the approach is that issuing a digital signature can be an expensive operation. (We present some measurements in Section 5.9.)

As mentioned in Section 5.6.2, a hidden constraint is represented as $\tilde{H}$ in an access right. Here, $\tilde{H}$ is a public key that is used to validate assurances signed with private key $\tilde{H}^{-1}$ saying that hidden constraint $\tilde{H}$ is satisfied. The issuer of an access right should generate $\tilde{H}$ (and $\tilde{H}^{-1}$) itself; $\tilde{H}$ should not correspond to the public key of the service providing the constraint information. To avoid information gathering based on correlation, $\tilde{H}$ should not be re-used in different access rights.

The constraint specification referred to by $\tilde{H}$ consists of the following parts:

Constraint definition. This part lists constraint information (e.g., $B.x$) and a set of permitted values.

Signing key. The signing key corresponds to private key $\tilde{H}^{-1}$. It is encrypted with the public key of the constraint service, S, that provides information $B.x$.[2] The issuer of an access right and of a constraint specification chooses S and thereby decides about the mapping from constraint information $B.x$ to constraint service S.

Validation key. The validation key corresponds to public key $\tilde{H}$.

Public key of service. This part lists the public key of constraint service S.

Integrity data. This data ensures the integrity of the constraint specification. We use a cryptographic hash of the constraint specification (excluding signing key and integrity data) and encrypt this hash together with the signing key.

This constraint specification and an access right containing reference $\tilde{H}$ to it are used as follows: Their issuer gives both of them to a client. When building a proof of access, the client retrieves the identity of the constraint service, S, from the specification and gives the constraint specification to S. The service ensures that the current value of $B.x$ corresponds

[2] We use an AES-based hybrid encryption scheme and HMAC for integrity checking.

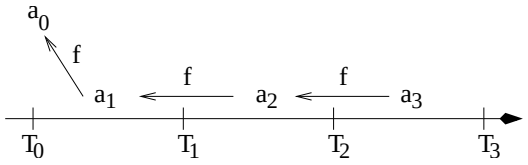


Figure 5.7: One-way chain. Each value in the chain, a_i, is associated with a time frame. Value a_0 validates values in the chain.

to one of the permitted values. It then decrypts the ciphertext in the specification to get $\tilde{H}^{-1}$ and to ensure that the specification has not been tampered with. It uses $\tilde{H}^{-1}$ to issue an assurance in the form of a digital certificate. The assurance consists of the validation key, $\tilde{H}$, signed with signing key $\tilde{H}^{-1}$ (i.e., the assurance expresses $\tilde{H}$ **says** $C \stackrel{\tilde{H}}{\Longrightarrow} \tilde{H}$).

The client sends the access right together with the assurance to the primary service, which validates the signature of the access right. For reference $\tilde{H}$ included in the access right, the service ensures that there is an assurance covering $\tilde{H}$ and signed with $\tilde{H}^{-1}$.

One-way Chains

Our second option does not require the generation of a potentially expensive digital signature when issuing an assurance. Instead, it is based on one-way chains. A one-way chain consists of a seed value a_n and repeated applications of a public one-way function, f, to this seed, or $a_{i-1} = f(a_i)$ for $0 < i \leq n$.

For each hidden constraint, the issuer of an access right chooses n and a_n and computes a_0. The issuer also picks a time in the future, T_0, and a time interval, ΔT. In our formal model, the tuple $\langle n, a_0, T_0, \Delta T \rangle$ represents $\tilde{H}$ and a_n represents $\tilde{H}^{-1}$. Given a tuple, it is possible to compute T_i with $T_i = T_{i-1} + \Delta T$ for $0 < i < n$. We associate a_i with time frame $[T_{i-1}, T_i]$ (see Figure 5.7). If a constraint is fulfilled in time frame $[T_{i-1}, T_i]$, we want a constraint service to release the associated value, a_i. If the primary service manages to recompute a_0 based on a_i, it knows that the constraint is satisfied. Based on the property of one-way chains, given a_i for time frame $[T_{i-1}, T_i]$, it is not possible to compute a_j for future time frames $[T_{j-1}, T_j]$ ($j > i$).

This approach has the advantage that it is less expensive for a constraint service. In particular, there are cryptographic hash functions (e.g., SHA-1) that can be used for implementing f and that are cheap to compute. Another advantage is that the approach allows trading off performance vs. convenience when deciding about the length of a one-way

chain. A one-way chains covers only a limited time period. After this time period, a new chain must be created and the access right using the chain needs to be re-issued. One the one hand, longer chains require fewer updates of access rights. On the other hand, longer chains make it more expensive for a constraint service or a primary service to compute values in the chain. (To avoid denial-of-service attacks, services can force issuers of access rights to limit the length of chains by limiting the number of steps that they are willing to compute.)

The approach has the drawback that it is the issuer of an access right that chooses the length of the time interval, ΔT, during which it expects a constraint to remain satisfied. A constraint service cannot incorporate its own domain-specific knowledge. However, for certain types of information, this limitation is not a problem. For example, for access rights constrained to location information, an issuer can choose an interval of a few minutes. Since the speed at which individuals move is finite, the error in location remains limited.

For one-way chains, the constraint specification looks as follows:

Constraint definition. This part is identical to the certificate-based case.

Seed. There is the seed of the one-way chain, a_n. It is encrypted with the public key of the constraint service, S, that provides information $B.x$.

Validation tuple. The validation tuple corresponds to the tuple $\langle n, a_0, T_0, \Delta T \rangle$.

Public key of service. This part lists the public key of the constraint service, S.

Integrity data. This part is identical to the certificate-based case.

When receiving a constraint specification from a client, a constraint service ensures that the constraint is satisfied and decrypts the ciphertext to get a_n and to detect tampering attacks. Based on the validation tuple, it determines the number of applications of f to a_n that are required to get a_i for the current time frame. Value a_i serves as assurance and is given to the client. The client forwards the value to the primary service. From the tuple $\langle n, a_0, T_0, \Delta T \rangle$ and the current time, this service figures out the number of applications of f to a_i required to get a_0. If the service computes a wrong value, either the client is cheating or the current time frame has moved since the constraint service computed a_i. (We assume reasonably synchronized clocks, depending on the granularity of ΔT.) In the latter case, the client will issue another request.

When the primary service manages to recompute a_0 from a_i, it can cache a_i and use the cached value for validating future requests that exploit the same access right. In this

way, a service does not have to recompute the chain from a_i to a_0 for each request. (Hu et al. [70] discuss techniques that allow a constraint service to speed up computation of a_i.)

Both the approach based on one-way chains and the one based on digital certificates require a constraint service to perform an asymmetric decryption operation, which can be expensive. However, it is possible for a constraint service to cache decrypted values. In this way, when a service is asked to issue an assurance for the same constraint multiple times, it needs to perform a decryption operation only for the first request.

5.8.3 Digital Certificates

In our architecture, we rely on extended SPKI/SDSI certificates [47] for expressing various statements. In particular, for non-hidden constraints, a principal issues a single certificate that expresses both an access right (i.e., Statement (5.1)) and a mapping from constraint information to a constraint service (i.e., Statement (5.4)). We combine access rights and mappings for simplicity reasons. If required, it is possible to design a more flexible scheme where a constraint in an access right only refers to a mapping, but does not directly include it. For the same reasons, we do not require a principal to list the range of values that the constraint information can have. We also rely on certificates for expressing assurances (i.e., Statement (5.6)), except for the approach based on one-way chains. We give three example statements in Figure 5.8.

In the first example, Carol grants Alice access to her calendar information if Alice is in Wean Hall 4103 and if an additional, hidden constraint is fulfilled. The non-hidden constraint should be validated by the constraint service with the given public key. The second and third example show an assurance for the non-hidden and hidden constraint, respectively.

It looks possible to implement constraints with threshold constructs, as explained in Section 3.10. Namely, we could implement a non-hidden constraint as a hidden constraint, and we could implement hidden constraints with threshold constructs. However, the latter step fails due to the following reason: For threshold constructs, a certificate lists multiple principals in the **subject** entry. For a principal to exploit such a certificate, (at least) a given number, k, of the listed principals must issue (maybe indirectly) further certificates that have this principal as their subject. For the certificate shown in Figure 5.8, the subject could list both Alice and the constraint validator and require $k = 2$. Therefore, Alice could use this certificate only if the validator issued another certificate to Alice. The problem with this approach is that it expects all certificates to cover the same kind of statement. However, in our scenario, the first certificate grants access to information, whereas the

```
(cert
   (issuer (pub_key:carol))
   (subject (pub_key:alice))
   (permission
      (information (pub_key:carol) carol calendar_entry))
   (tag
      (nonhidden-constraint
         (information (pub_key:alice) alice location)
         (''Wean Hall 4103'')
         (pub_key:service))
      (hidden-constraint
         (pub_key:validator))))

(assurance
   (issuer (pub_key:service))
   (nonhidden-constraint
      (information (pub_key:alice) alice location)
      (''Wean Hall 4103'')
      (pub_key:service)))

(assurance
   (issuer (pub_key:validator)))
```

Figure 5.8: Extended SPKI/SDSI certificates. We show an access right with constraints and two assurances. Signatures and lifetimes are omitted.

second one would state satisfaction of a constraint.

5.9 Performance Analysis

We now present a performance analysis of our distributed access-control architecture with support for constraints. We run our measurements on an unloaded Pentium IV/2.5 GHz with 1.5 GB of memory, Linux 2.4.20, and Java 1.4.2. Our asymmetric cryptographic operations employ 1024 bit RSA keys. We use SHA-1 for building one-way chains and authenticating constraint specifications. (Our machine can compute about 330,000 Java-based hashes/sec.) An experiment is run 100 times. We report the mean and standard deviation (in parentheses).

We study the cost of access control for the scenario where Carol grants Alice access to her calendar information under different constraints. In the first experiment, Carol grants access only if she is currently in her office. Carol does not hide this constraint. In the second experiment, Carol grants access only if Alice is currently in her office. Carol hides this constraint. If Carol did not hide the constraint, Alice would have to reveal her location to the calendar service, which she might have never used before and thus has no trust relationship with. In this experiment, Carol defines a hidden constraint based on certificates. The third experiment is identical to the second one, but the constraint service caches decrypted signing keys. In the fourth and fifth experiment, we repeat the second and third experiment, but the hidden constraint is based on a one-way chain, instead of certificates.

We assume that Alice has an unconstrained access right to Carol's location. In addition, the calendar service has an access right to Carol's location information, and Alice is in the possession of this access right.

The implementation builds an access-rights graph for Carol's calendar information in terms of Alice's access rights. In the first experiment, the implementation learns that it must retrieves an assurance for Carol's location before accessing the calendar information. It exploits Alice's access right to retrieve this assurance from the location service. (The location service fingers Carol's desktop computer and determines Carol's location based on her activity.) Next, the implementation ensures that the calendar service is authorized to access Carol's location information using the access right granted to the service. Similarly, Carol must have access to her location information, which is straightforward to validate. Finally, the implementation combines the assurance received from the location service and Alice's access right to the calendar information in a proof of access and sends the proof to the calendar service, which is a centralized calendar system running Oracle

CorporateTime.

In the other experiments, the implementation must retrieve an assurance for Alice's location instead of Carol's location. Furthermore, the implementation does not have to verify that the calendar service has access to Alice's location, since the constraint is hidden from the service. The implementation still needs to ensure that Carol as the issuer of the access right with Alice's location information as constraint information can access this information. The implementation can use an access right previously issued by Alice for this purpose, or it could ask Alice whether she wants to release her location to Carol. Finally, the proof of access is built in the same way as in the first experiment.

In the first experiment, the mean response time experienced by the client is 463 ms (26 ms). Detailed results are in Table 5.2. About 25% of the cost is due to retrieving an assurance from the constraint service. Most of this overhead is caused by setting up an SSL connection, which requires two costly RSA decryption/signing operations (about 16 ms each), and by acquiring the location information. The cost for issuing an assurance corresponds to the cost of generating a digital signature. Access control takes only a few milliseconds, the main cost is checking a signature.

Constraint processing has only limited influence on the primary service. In addition to checking the signature of the client's access right, it now also needs to validate the signature of the assurance. The main cost is due to SSL and the retrieval of the requested information. This SSL connection is more expensive than the first one. Closer inspection reveals that the additional delay is due to Java's garbage collection triggering during the setup.[3]

In the second experiment, we use a hidden constraint. The mean response time increases to 485 ms (14 ms). As shown in Table 5.2, the main cause for this increase is the larger cost for creating an assurance. Namely, the constraint service needs to decrypt the ciphertext in the constraint specification before it can issue an assurance. Deserialization cost also becomes larger since the constraint specification is now separate from the access right.

The third experiment is identical to the second one, but the constraint service caches decrypted ciphertexts. Therefore, the cost for issuing an assurance is reduced to the cost for generating a signature, as presented in Table 5.2.

For the next experiments, we use a hidden constraint based on one-way chains in Carol's access right. The chain has 2,016 steps and $\Delta T = 5$ minutes, that is, the chain covers a one-week period. During the lifetime of the one-way chain, the number of hash

[3]Choosing different amounts of memory allocations or using incremental garbage collection has no or negative influence on the results.

Entity	Step	Non-hidden		Hidden		Hidden, w/ caching	
		μ	(σ)	μ	(σ)	μ	(σ)
Client/constraint service	**SSL socket creation**	50	(3)	50	(3)	50	(3)
Constraint service	Deserialization	13	(2)	18	(2)	18	(3)
Constraint service	**Access control**	3	(1)	4	(2)	3	(2)
Constraint service	Retrieve location	37	(3)	38	(3)	38	(3)
Constraint service	**Issue assurance**	17	(1)	35	(1)	17	(1)
Client/primary service	**SSL socket creation**	92	(12)	96	(16)	96	(16)
Primary service	Deserialization	23	(4)	21	(7)	20	(2)
Primary service	**Access control**	5	(2)	5	(2)	5	(2)
Primary service	Retrieve calendar entry	202	(23)	204	(16)	201	(11)
	Total	463	(26)	485	(14)	469	(15)

Table 5.2: Client-response time. Mean and standard deviation of elapsed time for security operations (in bold) and for other, expensive operations using either non-hidden or hidden, certificate-based constraints [ms].

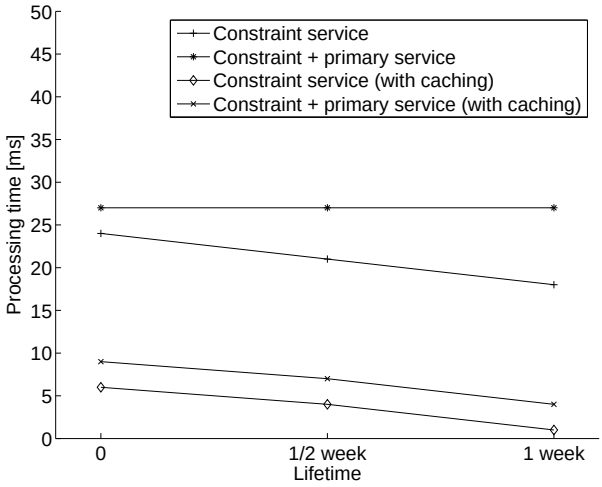


Figure 5.9: The curves show the processing time it takes the constraint service to issue an assurance and the cumulative cost of issuing this assurance and of the primary service running access control, varying over the lifetime of the hash chain

operations to be computed changes both for the constraint service and for the primary service. For example, at the beginning, the constraint service needs to compute 2,015 steps, after 1/2 week 1,008 steps, and after one week zero. This observation is confirmed by the first two curves in Figure 5.9, which show a) the constraint service's cost for issuing an assurance and b) the cumulative cost for issuing this assurance and for the primary service's cost for running access control. The cumulative cost remains constant. The results can be explained with the help of Figure 5.7: The number of hash operations required for computing the value associated with a time frame decreases over time, whereas the number of hash operations required for this value's validation increases, so the total number of operations remains constant. For this experiment, the mean client response time is about 468 ms (23 ms), which is less than in the case with hidden, certificate-based constraints (without caching).

Figure 5.9 also shows the processing time for the case where the constraint service caches decrypted ciphertexts and where the primary service caches one-way chain values that it receives from the client and that it successfully validates. The cumulative cost is minimal after one week, when the constraint service does not need to compute any hash operations and just returns the seed of the chain. For this case, the mean client-response time is 439 ms (14 ms).

5.10 Related Work

There are several areas of research that are relevant to our work. In particular, let us discuss context-sensitive access control in other pervasive computing environments, in role-based access control, and in access-control models designed with constraints in mind. We also take a look at inference-control research and automated trust negotiation

Multiple frameworks for pervasive computing environments support context-sensitive access control to confidential information [5, 38, 51, 93]. Al-Muhtadi et al. [5], Chen et al. [38], and Gandon and Sadeh [51] each employ centralized rule engines for running access control. None of them discusses whether and how they address information leaks caused by constraints. Minami and Kotz [93, 94] present a service-based access-control architecture that protects information in constraints. Access rights are public in their architecture. In case the primary service has an access right to the constraint information listed in the client's access right, the authors assume that there are no constraints in the service's access right. This limitation avoids the information leak mentioned in Section 5.4.1, where the client learns confidential knowledge about the constraint information in the service's access right.

Covington et al. [42, 43], Neumann and Strembeck [98], and Bacon et al. [10] add context awareness to role-based access control. The first two approaches make the assignment of a permission to a role conditional on the current context; the third one conditions role activations on the current context. The first two approaches assume centralized environments managed by a single entity; the third one supports distributed environments, where different entities manage roles. None of the approaches considers information leaks caused by context-sensitive constraints.

Classic access control models, such as mandatory access control, discretionary access control, or role-based access control, have no or very limited support for context-sensitive access rights to information. This limitation has been addressed in newer models, such as UCON_{ABC} [102] or GAA API [98]. In UCON_{ABC} (Usage Control based on Authorizations, oBligations, and Conditions), conditions can be used to support our notion of constraints. Similarly, the GAA API (Generic Authorization and Access-control API) supports various types of context-sensitive constraints. The authors of the two models do not discuss how their models address information leaks that context-sensitive constraints could cause.

McDaniel [91] discusses various evaluation issues for constraints in a distributed environment (e.g., asynchronous vs. synchronous evaluation). In addition, he lists security properties that such constraints should provide (e.g., non-repudiation) and reviews different implementation approaches. He does not discuss information leaks caused by constraints.

In information leaks due to confidential constraints, an attacker infers confidential knowledge by knowing the constraint specification in an access right and by observing a request exploiting this access right. Inference-control research [46, 3, 32] has looked at the problem of attackers inferring confidential knowledge by sending (sequences of) queries to a statistical database and correlating the received statistics. Statistics contain a trace of the data that have been used in the computation. Therefore, correlating multiple statistics might allow an attacker to infer confidential information about an individual. For example, by comparing the average salaries of two groups differing only by a single record, the attacker can deduce the salary of the individual whose record is in only one of the groups. As explained in Section 5.4, similar to information leaks due to confidential constraints, there can be positive and negative compromises and exact and partial compromises. Inference-control research has come up with restriction-based and perturbation-based techniques to protect against the inference of confidential information from statistical queries [32]. The former techniques restrict statistical queries by considering the size of the query set, by controlling overlapping between successive query sets, by keeping a history of queries, and by limiting the number of attributes involved in a query. The latter techniques intro-

duce some modification to the data stored in a database or to the requested statistics during the processing of a statistical query. Castano et al. [32] provide an overview of the various techniques.

Hidden constraints keep constraint specifications secret from clients or services. In automated trust negotiation [130], a client or a service keeps its credentials or access policy initially secret from a service or client, respectively. Here, after successfully completing (potentially multiple rounds of) negotiations between the client and the service, a party will typically know the other party's policy or credentials. For hidden constraints, the constraint specification must remain secret from a client or service throughout the processing of a query. A client or service might eventually be able to learn a constraint specification if it was authorized to access the constraint information listed in the specification and if it was able to observe the system. However, since the entity has access to the constraint information, this scenario does not result in an information leak. We provide a more detailed discussion of automated trust negotiation in Section 5.10.

5.11 Summary

We showed that context-sensitive constraints on access rights can lead to information leaks and that, depending on the access-control approach, information leaks must be addressed in different ways. We also introduced the concepts of access-rights graphs and hidden constraints. Access-rights graphs represent the conditions under which access should be granted and support easy detection of loops and conflicts. Hidden constraints avoid information leaks by hiding constraints from various entities.

We presented a client-based access-control architecture that supports context-sensitive constraints on access rights and that avoids information leaks. We discussed two implementations of hidden constraints, one based on digital signatures and another one based on one-way chains. Our sample implementation and its deployment demonstrate the feasibility of our approach. Its performance is competitive.

Our discussion revealed that access rights should be kept confidential, else information leaks due to confidential constraints become difficult to avoid. Furthermore, the types of constraints should be kept simple, otherwise running the algorithm to avoid information leaks can become complex. In particular, our discussion suggested that constraints should involve either a subject being granted an access right or a principal issuing an access right.

In the next chapter, we discuss how traditional client-based access control can result in information leaks when applied to pervasive computing and how to avoid these leaks.

Chapter 6

Obscured Proof-of-Access Descriptions

Traditional client-based access control assumes that either a client knows the nature of the required proof of access or that a service can inform the client of this nature. However, for complex information, this assumption no longer holds. If Carol's calendar entry states that she is meeting with Bob, Alice needs to present a proof of access consisting of access rights to Carol's location and activity information and to Bob's location and activity information. However, Alice does not know that Carol is meeting with Bob, so she does not know that she has to submit access rights to Bob's location and activity information. Similarly, the calendar service cannot tell Alice that she has to present access rights to Bob's information, since otherwise Alice could deduce that Carol is currently meeting with Bob, which could be an information leak. In this chapter, we present a solution for this problem in the context of pervasive computing, based on cryptography.

6.1 Overview

A naïve solution for a client that wants to access complex information, such as a calendar entry, is to have the client transmit a proof of access listing all potentially relevant access rights, such as all the client's access rights to location and activity information. This solution has privacy and bandwidth issues: a service can learn a lot about a client, and a client might have to transmit a lot of data.

Our proposed solution has a service return a description of the required proof of access to a client, but the service obscures the description such that the client can interpret the description only if the client has some secret knowledge [62]. The client has this secret knowledge only if it has access to the information listed in the proof description. In

particular, our contributions include

- a novel application of hierarchical identity-based encryption in the context of client-based access control,
- an extension of an existing scheme for hierarchical identity-based encryption to support multiple (hierarchical) constraints on access rights,
- a novel way for dealing with expiration in identity-based encryption, and
- the first implementation of a hierarchical identity-based encryption scheme, to the best of our knowledge.

We start by listing the requirements that an obscured description of a proof of access needs to fulfill (Section 6.2). Next, we introduce hierarchical identity-based encryption and discuss how we use it for obscuring proof descriptions in pervasive computing (Section 6.3). We explain how obscured proof descriptions interact with forwarded access rights, information relationships, and confidential constraints (Section 6.4). Finally, we have implemented our solution in a pervasive computing environment (Section 6.5).

We will provide an evaluation and a discussion of the relative strengths and weaknesses of our example implementation in the next chapter (i.e., Section 7.6), where we discuss another application scenario for hierarchical-identity-based encryption in pervasive computing.

6.2 Requirements

In client-based access control, if a client does not know the nature of the required proof of access, a service will give it a description of this proof, which lists the information for which the client needs to present access rights. However, when this proof description leaks confidential knowledge, the service must obscure it. Let us summarize the requirements for this case:

Indistinguishability. The service must obscure the description such that a client learns nothing about the owner of the information listed in the description, unless the client has some secret knowledge. The client has this secret knowledge only if it has an access right to the information.

Constraints. As mentioned in Section 2.2.2, access rights can have constraints on them. These constraints should also apply to a client's ability to interpret an obscured proof description. For example, when a client's access right to an information item expires, the client should no longer be able to interpret an obscured proof description asking for this item. To support this feature, each possible value of a constraint must require separate secret knowledge. For instance, a client's secret knowledge allowing interpretation of an obscured proof description on January 1 must not allow interpretation on January 2. This requirement leads to an increase in the secret knowledge to be managed by the client. The problem becomes worse when there are multiple constraints on an access right. We observe that many constraints are of a hierarchical nature. Therefore, we want a solution that supports hierarchical constraints. For example, if a client has secret knowledge for January, it can derive secret knowledge for January 1, January 2,... This feature simplifies management of the secret knowledge.

In the first part of this chapter, we limit ourselves to constraints whose current value is publicly known (e.g., current time). We discuss an extension to support constraints that involve confidential information (e.g., current location of the client or the queried individual) in Section 6.4.4.

Granularity awareness. Some information in pervasive computing (e.g., location information) is available at different granularities. To enforce that understanding an obscured proof description asking for fine-grained information does not allow one to understand an obscured proof description asking for coarse-grained information, the client's secret knowledge required for this understanding must be different for the two cases. Similar to constraints, a naïve implementation of this requirement can make management of the secret knowledge difficult. We can address this problem with a solution that supports hierarchical secret knowledge. In particular, if a client's secret knowledge allowed the client to interpret an obscured proof description asking for fine-grained information, the same knowledge should also allow the client to interpret an obscured proof description asking for coarse-grained information. This feature simplifies management of the secret knowledge.

Personalization. We want obscured proof descriptions to be personalized for a client. In this way, if one or multiple clients leaked their secret knowledge required for understanding an obscured proof description asking for some information, this knowledge would be useless to anyone else, and other clients being able to understand an obscured proof description asking for the same information would not have to acquire new secrets.

Asymmetry. As mentioned in Section 3.1, access rights are service independent. A service generating an obscured proof description listing specific information must not be able to interpret an obscured proof description listing the same information generated by another service (unless the former service has the required access rights). For example, an attacker could set up its own calendar service and have the service return obscured proof descriptions. However, the attacker must not be able to interpret obscured descriptions generated by another calendar service.

6.3 Obscured Proof-of-Access Descriptions

We want a client-based access-control architecture where obscured proof descriptions are simple to manage, aware of granularity, and constrainable. The solution also has to be asymmetric, provide indistinguishability, and be personalizable. Identity-based encryption (IBE) is a good fit for these requirements. It is asymmetric and provides indistinguishability. Since public keys are strings, proof management and personalization become simple. In addition, a hierarchical version of identity-based encryption lends itself to the implementation of hierarchical constraints and granularity awareness. Therefore, with the help of hierarchical identity-based encryption (HIBE), we can overcome the shortcomings of existing client-based access-control architectures for pervasive computing environments. In this section, we review HIBE and discuss how we extend it to build a solution that satisfies our requirements.

6.3.1 Hierarchical Identity-Based Encryption

In an IBE scheme, the public key of an individual is an arbitrary string, typically corresponding to her ID (e.g., her email address) [111]. The individual gets her private key from a third party, called a Private Key Generator (PKG). The third party also provides additional, public parameters required for the cryptographic operations. Boneh and Franklin [26] present one of the first practical IBE schemes. Based on this work, Gentry and Silverberg [54] introduce a HIBE scheme. In this scheme, a root PKG gives out private keys to sub PKGs, which in turn give out private keys to individuals in their domains (or further sub PKGs). The public key of an individual corresponds to the IDs associated with the root PKG, any sub PKGs on the path from the root PKG to the individual, and the individual. For encrypting messages, additional public parameters are required only from the root PKG.

The limited success of PKI has lead to the development of simpler public key infras-

tructures (e.g,. SPKI [47]), that do not require (hierarchical) certification authorities. In SPKI, a user's public key is her identity, and not her name, as certified by an authority. In our work, we pursue a similar approach. Instead of requiring the existence of a single hierarchical PKG infrastructure, we let each owner of information have its own PKG. The owner uses its PKG for managing access rights to its information. An owner can set up a hierarchical PKG infrastructure, where it controls both the root PKG and any sub PKGs. In this way, the owner will be able to establish granularity-aware access rights and hierarchical constraints (see Section 6.3.3). In the rest of this chapter, we refrain from using the term "PKG" and talk about owners instead.

Our architecture builds on the HIBE scheme proposed by Gentry and Silverberg. This scheme supports only a single hierarchy for a root PKG, which is too limiting for our application scenarios, where we might have multiple hierarchical constraints on access rights. Therefore, we extend the scheme to support multiple hierarchies.

A HIBE scheme has the advantage that it reduces the amount of required storage and the complexity of the key management. As we will see in Section 6.3.3, the public key of information corresponds directly to the name of the information. There is no need for storing a separate public key (obtained from a conventional cryptosystem such as RSA) for each information item. Maintaining the mappings from information to a key would also make access-right management more difficult. We discuss the advantages of a HIBE scheme in more detail in Section 6.4.1.

6.3.2 Basic Operations

Our solution employs four basic, randomized operations. We discuss these operations in this section and their application to client-based access control in the next section. Our operations are based on the operations introduced by Gentry and Silverberg [54], but we extend them to support multiple hierarchies. We give a detailed discussion, showing the exact cryptographic steps for each operation, in Appendix A.1. For readability reasons, we omit some of the parameters of the operations here.

In order to achieve indistinguishability, we assume that all the owners of information agree on a set of public parameters, $params$. The basic operations are $Root\_Setup()$, $Extract()$, $Encrypt()$, and $Decrypt()$.

- $Root\_Setup(params) \rightarrow Q_0$:
  An owner of information runs this operation in order to generate a master secret. In addition, the operation returns the owner's public key, Q_0.

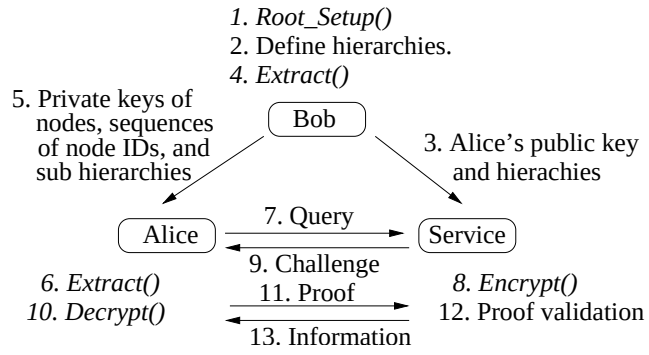


Figure 6.1: Architecture for client-based access control. The service sends a challenge to Alice. Upon resolving this challenge, Alice sends a proof of access to the service.

- $Extract(\langle ID_{i,1}, \ldots, ID_{i,t_i}\rangle, S_{i,t_i-1}, params) \rightarrow S_{i,t_i}$ with $t_i \geq 1$:
  This operation returns the private key, S_{i,t_i}, of a node at level t_i in hierarchy i. Unless $t_i = 1$, this key is derived from the private key of the parent node, S_{i,t_i-1}. If $t_i = 1$, this operation needs to be run by an owner, since it requires the owner's master secret. $\langle ID_{i,1}, \ldots, ID_{i,t_i}\rangle$ is the sequence of node IDs along the path from the root node of hierarchy i to the node in question.

- $Encrypt(\langle ID_{1,1}, \ldots, ID_{1,t_1}\rangle, \ldots, \langle ID_{h,1}, \ldots, ID_{h,t_h}\rangle, M, Q_0, params) \rightarrow C$: After choosing a node in each hierarchy, a service uses this operation to encrypt a message, M, using the nodes' public keys. For each of the h hierarchies, the operation accepts a sequence of node IDs, $\langle ID_{i,1}, \ldots, ID_{i,t_i}\rangle$, from the root node to the chosen node. The operation returns a ciphertext, C.

- $Decrypt(\langle S_{1,t_1}, \ldots, S_{h,t_h}\rangle, C, params) \rightarrow M$:
  A client uses this operation to decrypt a ciphertext, C. The operation requires the private key of each node chosen by the service in its call to $Encrypt()$ and the ciphertext.

6.3.3 Architecture

If Bob grants Alice an access right to information, he will also give her a personalized secret. When Alice receives an obscured proof description asking for this information from a service, this secret will allow her to interpret the description. In the rest of this chapter, we use the term *challenge* for such an obscured proof description. We keep management of

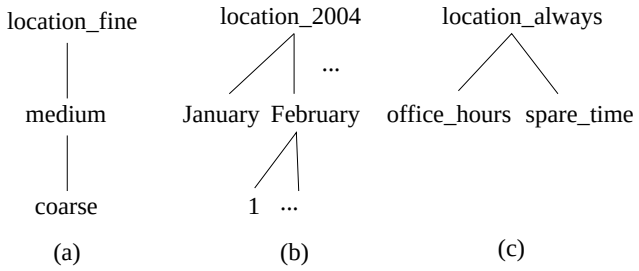


Figure 6.2: Hierarchies. Bob establishes hierarchies for his location information (a) and for each constraint (b, c).

the challenges simple by using the name of the information for generating a challenge for it. In our architecture, a challenge corresponds to a ciphertext/plaintext pair and a secret corresponds to a tuple of private keys enabling the decryption of ciphertexts. To support granularity-aware, constrainable challenges and secrets, Bob defines a set of hierarchies.

We give an overview of our client-based access-control architecture, extended with challenges, in Figure 6.1. There are three entities: an owner of personal information managing access rights to this information ("Bob), a service offering complex information that reveals some of the owner's information, and a client trying to access the complex information ("Alice). We now discuss the individual steps shown in Figure 6.1 in detail.

Setup. Bob runs $Root\_Setup()$ to set up his IBE scheme (1) and to retrieve his public key. He also establishes multiple hierarchies (2): He first defines a hierarchy resembling the granularity properties of information about him (*information hierarchy*). Figure 6.2 (a) gives an example hierarchy for location information. The rule for a hierarchy is that anyone who has access to information covered by a node should also have access to information covered by a child node. Bob then establishes another hierarchy for each of the constraints that he wants to include in his access rights to location information (*constraint hierarchies*). Figure 6.2 (b) shows a hierarchy that restricts the lifetime of access rights, and Figure 6.2 (c) presents a hierarchy for limiting access based on time of the day. (Non-hierarchical constraints are dealt with similarly; there, the hierarchy has only one level and lists all possible values.) The root node of each hierarchy includes the name of the information to ensure that, for example, a constraint granting unlimited access to location information cannot be used for getting unlimited access to medical information.

Bob then informs the service of his public key and his hierarchies (3). Since none of this knowledge is confidential, an authenticated communication channel suffices. Instead of defining his own hierarchies, Bob can exploit predefined hierarchies that the service is

already aware of. For example, we expect that there will be a widely accepted and shared hierarchy for location information.

To allow Bob to issue personalized secrets to clients, we have him personalize the information hierarchy by adding the identity of a client to the root node of the hierarchy. For example, for the hierarchy given in Figure 6.2 (a), the root node becomes "location_fine_Alice".[1] Since this personalization is done in the same way for each client, there is no need for Bob to submit each personalized information hierarchy to the service. To avoid collusion attacks between clients, Bob should also personalize each of his constraint hierarchies.

When issuing an access right to Alice, Bob also gives Alice a personalized secret, corresponding to the information in the access right and limited to the same constraints (5). In his information hierarchy, he chooses the node corresponding to the information to which he wants Alice to have access (e.g., "medium"). He then walks the path from the root node to this node. In particular, he keeps a sequence of node IDs and, for each node on the path, he calls $Extract()$ with the current sequence (e.g., $Extract(\langle$location_fine_Alice$\rangle$, null, $params) \rightarrow S_{1,1}$ and $Extract(\langle$location_fine_Alice, medium$\rangle, S_{1,1}, params) \rightarrow S_{1,2}$) (4). Ultimately, this process will return the private key of the chosen node. Similarly, for each type of constraint, he picks the appropriate node in the corresponding constraint hierarchy and derives the private key by repeated calls to $Extract()$. For each hierarchy, Bob will end up with a private key. The tuple of private keys returned by these calls serve as the secret.

Bob then gives the secret to Alice, together with the corresponding sequences of node IDs and the sub-hierarchies rooted in the chosen nodes (5). Transfer of the secret requires a secret communication channel.

Given the tuple of private keys and the sub-hierarchies from Bob, Alice can derive additional tuples of private keys for nodes in the sub-hierarchies by (repeatedly) calling $Extract()$ (6). For example, given the private key for $\langle$location_fine_Alice, medium$\rangle$ and the sub-hierarchy "coarse", Alice can extract the private key for $\langle$location_fine_Alice, medium, coarse$\rangle$. It is possible for Alice to delay this step until she receives a ciphertext.

Access Control. Alice issues a query to the service and fails to submit a proof of access (7). Since the requested information (e.g., calendar information) is complex, the service computes a challenge (8). In particular, the service calls $Encrypt()$ to encrypt a random plaintext, M. The public keys required for this operation come from the information and constraint hierarchies of the owner of the information for which the client needs to present an access right. The service locates the corresponding node in Bob's information

[1] In the actual implementation, Alice is identified by her public key.

hierarchy. The service then gathers the IDs of all the nodes along the path from the root node to this node. For example, if an access right to fine-grained information is required, the ID sequence is ⟨location_fine_Alice⟩. Similarly, for each of the constraint hierarchies, the service chooses the leaf node that contains the current value of the constraint and gathers the IDs along the path from the root node. The service then calls $Encrypt()$ with the gathered sequences of node IDs (e.g., $Encrypt(\langle$location_fine_Alice$\rangle,$ $\langle$location_2005_Alice,February, 2$\rangle, \langle$location_always_Alice, office_hours$\rangle, M, Q_0, params)$). Note that the public keys used for encryption correspond directly to the node IDs.

The plaintext, M, and the obtained ciphertext, C, serve as the challenge, and the service sends them to Alice (9). If the requested information covers multiple individuals, there will be multiple challenges. Sending a challenge to Alice requires only an authenticated communication channel, since a challenge is personalized to a client and useless to other clients (without knowing the corresponding personalized secret).

To resolve challenge (M, C), Alice needs to find a tuple of private keys that makes ciphertext C decrypt to plaintext M. In particular, Alice calls $Decrypt()$ for each of her (potentially derived) tuples of private keys given to her by Bob (and other individuals) (10). She stops when the returned plaintext is identical to M. We discuss ways to limit the search space in Section 6.3.4. If Alice successfully resolves the challenge(s), she will resubmit the query, together with the required proof of access (11). The service will validate the proof (12) and return the requested information (13). Steps (11) and (13) require a secret communication channel.

Discussion. The benefits of our architecture are secrets that are personalized and granularity aware and that support constraints. Because a challenge for information is based on the name of the information, challenges are simple to manage. As opposed to a previous approach for dealing with expiration [26], which makes the current date part of an ID, our approach does not require handing out separate private keys for each possible date.

A client resolves a challenge before submitting the required proof of access to a service. However, for some scenarios, this second step can be omitted since resolving the challenge(s) already gives the client all the information that it is asking for. For example, if the client asks for the people in a room, the client will require access to all these people's location information. The service thus sends a challenge for each person's location information to the client. After resolving these challenges, the client knows about all the people in the room and hence about all the originally requested information and can skip submission of a proof of access. An obvious question is why not skip this second step all the time and stop using proofs of access? In this model, the service would encrypt the requested information instead of a random plaintext (as suggested by Holt et al. [65]). We refrain from adapting this model because, as we will see in Section 7.6, the decryption

operation is expensive. We view obscured proof-of-access descriptions as a special case. For most queries, we expect clients to know what they need to deliver a proof of access for. Therefore, we do not place the burden of decrypting ciphertexts on them for every request to confidential information.

Security Analysis. The security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem. (Please refer to Appendix A.1 for details.) Given this assumption, Gentry and Silverberg [54] show that their HIBE scheme has adaptive chosen ciphertext security in the random oracle model. It is straightforward to adapt their proof for multiple hierarchies. Therefore, attackers cannot decrypt ciphertexts without having the required decryption key. This property, together with personalization, also makes decryption keys leaked by clients useless to an attacker (as long as the attacker does not manage to impersonate one of these clients). Since a tuple of private keys is bound to a particular type of information, an attacker cannot use this tuple for other information.

Internally, the $Encrypt()/Decrypt()$ operations compute a ciphertext/plaintext by hashing a computed value into the domain of the plaintext/ciphertext and by XORing the hashed value with the plaintext/ciphertext. Instead of using $Encrypt()/Decrypt()$ on a known plaintext-ciphertext pair, we could omit the hashing and XORing steps and directly use the computed value as a challenge. This approach relies on the hardness of the Decision Bilinear Diffie-Hellman problem. (Please refer to Appendix A.1 for details.) We choose the approach based on $Encrypt()/Decrypt$ since it allows us to use the same basic routines for obscuring proof descriptions and for encryption-based access control (see Chapter 7) and since the Bilinear Diffie-Hellman problem is at least as hard as the Decision Bilinear Diffie-Hellman problem.

When choosing a random plaintext, a service should choose one that is long enough to make the probability of the client seeing a false positive while resolving the challenge is small. (For a plaintext of length l, the probability of a false positive when using random tuples of private keys is $1/2^l$.) A false positive will make the client send an irrelevant access right to the service. If the access right contained private information, this information would leak to the service.

Ciphertexts are indistinguishable, meaning that it is not possible to learn from a ciphertext which owner's public key was used for producing this ciphertext. Holt et al. [65] prove this property for the scenario where all owners of information share the same set of public parameters, as assumed in our model. The ciphertext can reveal the nature of the information [2]. However, as explained in Section 6.3.4, a client already has this knowledge. The ciphertext can also reveal information about the number and depths of the hierarchies used for encryption. We can address this issue by limiting the number of constraint hierarchies and the depth of both information and constraint hierarchies that we allow owners of

information to define. If an owner defined fewer constraint hierarchies or hierarchies that are not as deep as the allowed maximum level, a service could exploit dummy hierarchies or dummy levels for encryption to guarantee that the obtained ciphertext always contains the same number of components.

If a service returned multiple challenges, the number of challenges could leak some information to a client. For example, this number could reveal the number of people a person is meeting with. We can avoid this leak by making the client prove to the service that it can access this information before returning challenges to the client. Alternatively, the service could limit the number of returned challenges and use dummy challenges if the actual number was smaller than this limit.

6.3.4 Limiting the Search Space

As discussed in the previous section, Alice will have to search through her tuples of private keys for a tuple that allows decryption. We discuss some optimization strategies for the search in this section.

We first concentrate on the scenario where the challenge returned by a service covers only a single individual, that is, Alice needs to find only one tuple of private keys. As described in Section 6.3.3, when an owner of information gives a tuple of private keys to Alice granting her access to information under some constraints, Alice can potentially derive additional tuples from this tuple. We argue that among the original tuple and the derived tuples, at most one tuple is of relevance for the search. For each constraint hierarchy, Alice knows the current value of the constraint and can throw out all the tuples that do not include the corresponding private key. Alice can also limit the search space for the information hierarchy since the service can inform Alice of the nature and the granularity of the information for which she needs to resolve a challenge. For example, it is well known that calendar information is composed of fine-grained location information, but not of medical information. Therefore, the service can inform Alice that a challenge involves fine-grained location information. In summary, for all tuples of private keys given to Alice by a single owner of information and all tuples derivable from these tuples, we expect at most one tuple to be relevant for a search. Overall, the number of tuples that Alice needs to search is at most one per owner of information.

If a service returned multiple challenges, Alice will have to locate multiple tuples of private keys. Therefore, Alice's search cost is proportional to the number of owners of information multiplied by the number of challenges. While this sounds expensive, Bradshaw et al. [29] present an optimization that requires the client to perform the most

	Personal hierarchy		Shared hierarchy	
	Public values	Private keys	Public values	Private keys
Conventional cryptosystem	$2n$	1	n	1
HIBE scheme	n	1	0	1

Table 6.1: Key-management demand. For a hierarchy of n nodes, we show the number of public values (including public keys) and private keys that an owner of information needs to define and give to a service and to a client, respectively.

expensive cryptographic operation in this search only once for each owner of information and not for each combination of an owner and a challenge.

6.4 Discussion

In this section, we compare our HIBE-based approach with alternative solutions. We also discuss how our solution deals with forwarded access rights, information relationships, and confidential constraints.

6.4.1 Identity-Based Encryption

IBE simplifies key management. For example, in an email system, IBE allows Bob to encrypt email to Alice simply by using her email address as public key. Bob does not need to contact Alice beforehand to acquire a separate public key. We seem to lose this advantage: Alice needs to inform a service of her hierarchies and her public key. However, as mentioned in Section 6.3.3, we do not expect each owner of information to define a set of hierarchies. Instead, there can be a shared set of hierarchies, which a service is aware of. In addition, we observe that a setup step is also necessary for IBE in an email system: First, IBE schemes require a set of public parameters for encryption. Bob must acquire these parameters before he can encrypt email for Alice. Second, Bob should ensure that the email address he is going to use to encrypt information destined for Alice really belongs to Alice. He should use this address only if he was given it directly by Alice (or a trusted third entity) in a setup step.

Instead of using a HIBE scheme, it is possible to make a conventional asymmetric cryptosystem, such as ElGamal or RSA, hierarchy aware. That is, given the private key of a node, it is possible to derive the private key of a child node. For ElGamal, we choose a

random value for the private key of the root node and build a hash tree rooted at this node. For example, in a binary tree, the private keys of the left and right child node of a node with private key x are $H(0||x)$ and $H(1||x)$, respectively. ($H()$ is a cryptographic hash function and $||$ denotes concatenation.) An owner of information builds this tree such that its structure is identical to the structure of the owner's information (or constraint) hierarchy. In ElGamal, the public key of a node is now computed as $y = g^x \bmod p$, where g and p are publicly known and x is the node's private key. For RSA, Ray et al. [107] present a hierarchy-aware version. The drawback of these solutions is their increased demand in key management and transfer. We summarize this demand in Table 6.1. (Both conventional and HIBE schemes typically also require storage and transfer of a constant amount of additional information, which is not shown in the table.) If an owner of information defined a set of hierarchies, the owner will have to transfer at least the ID of each node to a service in order to inform the service of the node's meaning, regardless of the employed hierarchical cryptosystem. For a HIBE scheme, only this ID is required. For a conventional cryptosystem, a separate public key needs to be generated and transferred for each node. If an owner of information uses a shared information or constraint hierarchy and employs a conventional cryptosystem, it will still have to generate a set of public keys for all the nodes in the shared hierarchy and submit these values to individual information services. There is no such need for a HIBE scheme. Finally, whereas it is possible to achieve indistinguishability with an ElGamal-based solution, it is an open problem whether this is possible for an RSA-based solution [33].

While not being part of our security model, a deployed system needs to be able to deal with key revocation, attackers learning private keys or, worse, the compromise of an owner of information's master secret. We can exploit mechanisms proposed earlier [26, 114] for this purpose, that is, adding a salt to the naming scheme, including a time-to-live value with configuration information, and storing secrets in a distributed way. When compared with conventional asymmetric cryptosystems, key revocation becomes easier in our HIBE scheme. In particular, an owner of information can associate the same salt with the root node of each of the owner's hierarchies. Upon key revocation, in terms of public values, only this salt needs to be re-distributed. For a conventional cryptosystem, the public key associated with each node needs to be redistributed.

Our access-control mechanism is also suitable to environments other than pervasive computing. The mechanism targets scenarios where multiple information services, run by different organizations, need to distribute the same kind of information to the same set of clients. Another deployment option is in the context of medical information, where multiple hospitals, run by different HMOs, grant the same researchers access to the same kind of confidential statistical information gathered in a hospital. Moreover, an extended ver-

sion of our solution allows for secret handshakes [12] with flexible, expiring membership credentials. In a secret handshake, members of the same group authenticate each other secretly such that non-members cannot recognize group members and non-members cannot perform the handshake. Traditional solutions [12, 33] require explicit revocation of membership certificates or have limited flexibility in terms of expiration (e.g., all credentials are valid for exactly one day).

As we will see in Section 7.6, our proposed HIBE scheme can be expensive in terms of performance. This cost could become a problem in a pervasive computing environment, where clients might employ computationally weak devices for accessing information (e.g., a cellphone). A common architecture for such environments is to have agents perform tasks on behalf of clients [38, 51]. We could have this agent decrypt information for its client. For performance and availability reasons, it makes sense to run this agent on a more powerful processing platform and to run only a lightweight proxy on a client's personal device.

6.4.2 Forwarded Access Rights

Our access-control architecture introduced in Chapter 2 supports forwarding of access rights, that is, a client that is in the possession of an access right can forward this access right to other entities by issuing additional access rights (i.e., digital certificates). However, our solution for obscured proof-of-access descriptions does not support forwarding, that is, a client that is given a tuple of private keys by an owner of information, together with an access right, cannot forward this tuple to other entities. In particular, giving a copy of the tuple to other entities does not help these entities since tuples are personalized. Alternatively, the client cannot generate tuples specific to these entities since this generation requires the master secret created by the owner of the information, which is not available to the client.

This limitation seems difficult to overcome, since there are conflicting interests. On the one hand, we want the secret knowledge that is given by an owner of information to a client to be personalized so that leakage of this knowledge does not affect other clients. On the other hand, we want a client to be able to exploit its secret knowledge for generating new secret knowledge specific to other clients.

In our current solution, there are two options for dealing with forwarded access rights. The first option is to give up personalization, that is, all clients authorized to access a particular type of information are given the same secret. This approach allows clients to copy this secret when forwarding an access right. As mentioned in Section 6.2, the

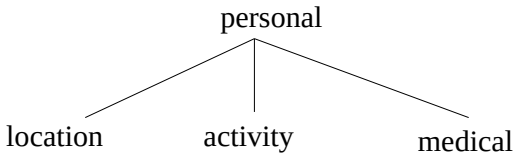


Figure 6.3: Bundling-based information hierarchy. The hierarchy represents bundling-based information relationships.

drawback of this approach is that all clients need to be issued a new secret when the secret leaks. The second option is to have a client that is granted a forwarded access right submit all potentially relevant access rights to a service. As mentioned in Section 6.1, this approach has privacy and bandwidth issues.

6.4.3 Information Relationships

As mentioned in Section 3.3.3, granularity-based information relationships are a special case of bundling-based relationships. Since information hierarchies represent granularity properties of information and thus granularity-based relationships, it looks possible to also model bundling-based information relationships with information hierarchies. For example, Figure 6.3 illustrates how an information hierarchy can represent bundling-based information relationships, where an individual's location, activity, and medical information are bundled in her personal information. While it is possible to express bundling-based information relationships with information hierarchies, there are three potential obstacles that we need to overcome.

First, we must ensure that the underlying HIBE scheme is secure against collusion. For example, for the information hierarchy given in Figure 6.3, assume that Alice has the tuple of private keys for (⟨personal_Alice, location⟩, ⟨personal_2005_Alice, January⟩) and (⟨personal_Alice, medical⟩, ⟨personal_2005_Alice, February⟩). Given the two tuples, Alice should not be able to determine the tuple of private keys for (⟨personal_Alice, location⟩, ⟨personal_2005_Alice, February⟩) and (⟨personal_Alice, medical⟩, ⟨personal_2005, January⟩). Our HIBE scheme is not secure against collusion. Yao et al. [133] propose a collusion-resistant HIBE scheme, which we could also adopt. However, the complexity of the $Encrypt()$ and $Decrypt()$ operations in their scheme is $\mathcal{O}(n^m)$, where n is the depth of a hierarchy and m is the number of hierarchies. As we will see in Section 7.6, the complexity of the operations in our scheme is $\mathcal{O}(mn)$.

Second, information hierarchies support only tree-based hierarchies, that is, informa-

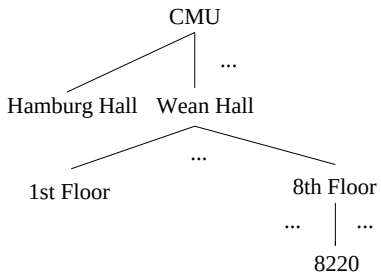


Figure 6.4: Location hierarchy. Hierarchy for location-dependent access rights.

tion can be bundled only in one type of information. For example, it is not possible to bundle location information in more than one other type of information. Tree-based hierarchies could be sufficient for many individuals. Alternatively, if information needs to be bundled in multiple types of information, we can establish multiple information hierarchies for the bundled information. A service would then issue a separate challenge for each information hierarchy.

Third, information hierarchies assume that the information in all nodes of a hierarchy is owned by the same entity. This limitation is probably not severe for many individuals.

6.4.4 Confidential Constraints

We have assumed that the constraints on an access right involve non-confidential information, that is, information whose current value is publicly known. Let us now look at constraints that involve confidential information.

One such constraint is the location of a client. For example, Bob constrains an access right granted to Alice based on her location. Ideally, we can treat confidential constraints in the same way as non-confidential constraints. For example, in order to support a constraint based on Alice's location, we could define another constraint hierarchy corresponding to Alice's possible locations, such as the one shown in Figure 6.4, and treat this hierarchy in the same way as the information and constraint hierarchies discussed so far. Representing the constraint in a hierarchy allows Bob to easily grant an access right that is constrained based on an entire building/area, without having to enumerate each place contained in the building/area in the constraint.

However, this approach is insufficient due to the following reasons: First, Alice should be able to perform a decryption operation only if she actually is at the listed location.

Though Bob indicates this location in his access right granted to Alice, he must not give the corresponding private key to Alice when issuing the access right. Instead, Alice should be given this key only once she actually is at this location, potentially by a third entity, as determined by Bob. Second, in the given approach, Alice's decryption key for a particular location remains identical over time. However, Alice's location changes over time. Correspondingly, the decryption key for a particular location should change such that Alice cannot re-use the key once she leaves this location (or, at most, she can re-use it only within a short time frame). Third, the service uses the current value of the constraint (i.e., Alice's current location) as a public key when encrypting a plaintext. However, this value is not public, and the service might not know it. In summary, we cannot treat confidential constraints in the same way as non-confidential constraints.

As an alternative, it looks possible to design a solution based on an "inverted" location hierarchy. In particular, in this solution, the service knows Alice's access right and uses the corresponding public key for encrypting a plaintext. For example, if Alice's access right is constrained to her being in "CMU/Wean Hall", the service could use this value as a public key. If Alice is at any location in Wean Hall (e.g., "CMU/Wean Hall/8th Floor/8220"), she is given the private key for this location. The approach should then allow Alice to derive the private key for "CMU/Wean Hall". However, this approach has the drawback that it requires the service to know about each individual's access right.

The two discussed solutions fail because they assume that the service is aware of the current value or the nature of a constraint in Alice's access right. Instead, we now propose a solution in which the service is oblivious to these two items.

Let us assume that Bob hands out access rights to his location information and that he wants to constrain Alice's access based on her location. He first decides about the entity that is responsible for guaranteeing satisfaction of the constraint. In our example, Dave is this entity. Bob then informs Dave of the nature of the constraint. Note that this specification can be client-specific, that is, Bob could require Alice to be at a location different from the location required for another client. In addition, Bob informs the service of Dave's public key, Q_0^{Dave}, and tells the service how long he expects a constraint to remain valid. Finally, Bob lets Alice know about the constraint specification.

When Alice contacts the service, the service uses an extended version of the $Encrypt()$ operation, which supports multiple individuals' public keys, to encrypt a random plaintext. For this encryption, the service exploits the already discussed public keys for nodes in Bob's information and constraint hierarchies, where the constraint hierarchies cover non-confidential constraints. In addition, the service exploits a public key covering the confidential constraint, where the corresponding private key is generated by Dave. Namely, this public key is a concatenation of the type of information in the client's access right

("location"), the ID of the client ("Alice"), the ID of the owner of the information in the access right ("Bob"), and the time frame during which the constraint is expected to remain valid (e.g., "2005/05/25/14:00:00 EDT - 2005/05/25/14:05:00 EDT"). Note that the public key does not list the nature of the constraint or the required value; we assume that Dave provides only one constraint per type of information, per client, and per owner. The service then uses these public keys to execute $Encrypt()$. For example, the service calls $Encrypt(\langle$location_fine_Alice$\rangle, \langle$location_2005_Alice,February, 2$\rangle,$
$\langle$location_always_Alice,office_hours$\rangle,$
location_Alice_Bob_2005/05/25/14:00:00EDT-2005/05/25/14:05:00EDT$, M, Q_0^{Bob}, Q_0^{Dave},$
$params)$.

In order to decrypt the ciphertext, Alice needs to retrieve a private key from Dave. Dave maintains a mapping from the type of information in an access right ("location"), the ID of a client ("Alice"), and the ID of the owner of the information in the access right ("Bob") to a constraint specification listed in the access right. After looking up this constraint specification, Dave needs to ensure that no information leaks to Bob or to Alice, that is, Bob and Alice need to have access to the constraint information. Then, Dave validates that the constraint is satisfied and makes sure that Alice is at the location required by Bob. If this validation is successful, Dave will compute the corresponding private key (e.g., $Extract($location_Alice_Bob_2005/05/25/14:00:00EDT-2005/05/25/14:05:00EDT$, s_0^{Dave},$ $params)$, where s_0^{Dave} is Dave's master secret, and he will give the key to Alice. This private key (and private keys obtained from Bob) will allow Alice to decrypt the ciphertext received from the service.

6.5 Prototype Implementation

Identity-based cryptography has been used for different types of applications, such as searchable audit logs [126] or secure email and IPsec [114]. All these applications, including our proposed one, exploit the Boneh and Franklin IBE scheme [26]. While there are other schemes (e.g., by Boneh and Boyen [24], Boneh et al. [25], Cocks [41], Waters [125], and Yao et al. [133]), we choose this scheme because of the existence of publicly available implementations [56, 89].

Because the Aura environment is mostly Java, we implemented our HIBE scheme in Java. We ported a C implementation of IBE [56] to Java and added support for hierarchies. We employ a hybrid encryption scheme, that is, we symmetrically encrypt information with a session key and encrypt only this session key with $Encrypt()$.

We also implemented a calendar service that generates obscured proof descriptions.

Bob provides the public parameters of his identity-based cryptographic scheme and his hierarchies in SPKI/SDSI "auto-certificates" [48], whose purpose is to make information about their issuer available in an authentic way. Bob also uses auto-certificates for handing out tuples of private keys. There is a command line tool for issuing certificates, setting up IBE schemes, and extracting private keys.

We use SSL for communication between entities, which gives us authentication of peers and confidentiality and integrity of the transmitted data. Strictly speaking, we do not require confidentiality of a challenge in client-based access control. Server authentication and query confidentiality and integrity are required to deal with attackers listening, modifying, or injecting traffic. We require data integrity since our IBE implementation is only semantically secure, but does not provide chosen-ciphertext security. Finally, there has to be client authentication. We decided against implementing the required features, together with IBE, in a protocol of our own, since, as history has shown, correctly implementing protocols is hard. SSL has been well researched, and the overhead caused by the redundant, symmetric encryption is low.

We provide an evaluation of our implementation in Section 7.6.

6.6 Related Work

There are several areas of research that are relevant to our work. Let us review this research and discuss whether and how we can exploit it for obscured proof-of-access descriptions.

Holt et al. [65] introduce *Hidden Credentials*, where a service requires a client to be in the possession of a set of credentials in order to access confidential information, but the service does not make the client reveal these credentials. The solution also supports confidential access policies, where the service keeps its access policy listing the required credentials secret from the client. Similar to our work, Holt et al.'s work is based on the Boneh and Franklin IBE scheme. The service encrypts its confidential information, where the public key corresponds to the string composed of the content of the required credential and the identity of the client. If multiple credentials are required, an item is encrypted with each credential using onion encryption. Alternatively, if there is a choice in credentials, an item is encrypted multiple times, once for each possible credential. A client's credential is specific to the client and corresponds to an IBE private key. If the service's access policy is confidential, the service does not inform the client of the credential(s) required for decryption. The client then needs to find the right credential(s) in its set. In our architecture, due to reasons outlined in Section 6.3.3, we do not have a service encrypt information for client-based access control. Holt et al.'s and our architecture were developed indepen-

dently from each other. Holt et al. do not investigate constraints on credentials (i.e., access rights) and expiration of credentials.

Bradshaw et al. [29] extend Holt et al.'s architecture to support complex access policies, expressed as monotonic boolean functions, such that the structure of a policy remains concealed from an unauthorized client. They apply a secret splitting system and split a secret according to the structure of the policy. To prevent leakage of information about the number of required credentials, Bradshaw et al. use dummy shares and have a service return a constant number of (encrypted) secret shares. For obscured proof-of-access descriptions, the structure of an obscured description is known to the client (i.e., it is a series of AND operations) and we do not need to hide the structure.

Frikken et al. [50] also extend Holt et al.'s architecture to conceal complex policies from a client. In addition, their solution prevents a client from learning which of the client's credentials gained the client access. Frikken et al. only outline their solution and do not provide any details.

Persiano and Visconti [103] present an extension of SSL called "Secure and Private Socket Layer (SPSL)". The extension allows a service to present a set of X.509 certificates to a client. The client will be granted access if it knows the private key associated with at least one of the certificates, without revealing which one. For obscured proof-of-access descriptions, there is no choice in the permissible certificates.

Balfanz et al. [12] introduce the concept of *secret handshakes*, which allows two entities to authenticate each other only if both entities are members of the same group. Nonmembers cannot recognize such a handshake and cannot perform the handshake. An entity can require that the other entity has a particular role in the group for the handshake to succeed. Balfanz et al. present an implementation based on pairing-based cryptography. Castelluccia et al. [33] present an implementation based on the Computational Diffie-Hellmann assumption. The presented solutions assume that both entities have membership credentials issued by the same organization. Our architecture (and Holt et al.'s architecture) can be easily extended to support secret handshakes for scenarios where the two entities require each other to be members of different groups, that is, where the two entities have membership credentials issued by different organizations.

Li et al. [86] propose *Oblivious Signature-Based Envelope (OSBE)*. OSBE assumes that a client and a service agree on the content of a credential required by the client for accessing confidential information provided by the service and on the centralized entity issuing such credentials. (For example, the service informs the client of the content and of the authority at the beginning of an interaction.) The service then encrypts confidential information such that the client can decrypt the ciphertext only if the client has a signature

for the required credential, as issued by the centralized entity. Li et al. present implementations based on RSA and identity-based signatures. In our architecture, the service cannot inform the client of the content of the required credential. Also, the two parties do not agree on a centralized entity.

Brands [30] introduces several restrictive blind issuing protocols and showing protocols with selective disclosure for digital certificates. The protocols allow a client that is issued a certificate by a centralized entity to selectively reveal some of the client's properties listed in the certificate to a service. That is, the client does not have to reveal the entire certificate to the service, which prevents the service from learning unnecessary properties about the client and from tracking the client. Brands presents implementations based on RSA and discrete logarithms. Chaum [35] introduces a pseudonym system, where a client can transform a credential such that the client can show the credential to a service under a pseudonym. The solution also supports selective disclosure of the client's properties listed in the credential. Our solution does not require the client to reveal credentials to the service. Similar to OSBE, the two alternative solutions assume that the service can inform the client of the content of the required certificate/credential and that the two parties agree on a centralized entity. These assumptions do not hold in our scenario.

Automated trust negotiation [130] explores the establishment of trust between strangers based on potentially sensitive access policies and credentials. Seamons et al. [110] have a service represent an access policy as a directed, acyclic graph, where nodes represent parts of an access policy and edges determine the order in which the access policy can be revealed to a client. For access control, the service traverses the graph and tells the client in a stepwise-fashion about the contents of a node and what credential the client needs to present next. Yu and Winslett [134] extend Seamons et al.'s approach by separating policy satisfaction from policy disclosure, that is, the conditions under which the contents of a node can be revealed are different from the conditions for its satisfaction. For both approaches, the client can also build policy graphs, defining its willingness to reveal a credential to the service. For this case, Winsborough and Li [129] observe that if the client asks the service for a credential, the client probably has the credential requested by the service, which could be an information leak. To avoid this leak, the authors suggest that the client's reaction should be independent of the fact of whether it actually has the credential. The client defines this reaction in an "Ack policy". It is possible to look at obscured proof-of-access descriptions as an instance of automated trust negotiation. We assume that a client's access rights are not confidential, so there is no need for Ack policies. Yu and Winslett propose two strategies for exchanging access rights; neither of them is satisfactory for our scenario. In the first strategy, the client transmits all its access rights to the service, even if they are not required. This strategy has bandwidth issues. In addi-

tion, since it can lead to the disclosure of irrelevant access rights, it has privacy issues. In the second strategy, the client transmits only access rights that the service explicitly asks for by revealing (parts of) its access policy. However, this strategy fails if access rights whose corresponding access policy the service cannot reveal are required, as it is the case for obscured proof descriptions.

Keys with a limited lifetime have been proposed for IBE before [26]. The proposal makes the lifetime of a key an explicit part of the key's ID. It assumes that all keys have the same lifetime (e.g, all keys are valid for a year). In our approach, different keys can have different lifetimes.

6.7 Summary

When running client-based access control to confidential complex information, we need to avoid the situation where a client can infer confidential information from the description of a required proof of access. We showed how hierarchical identity-based encryption can be employed to obscure this description such that only a client that is authorized to access the information listed in the description can interpret the description. Our approach supports information of different granularity levels and constraints on access rights. We incorporated our proposed solution into a client-based access-control architecture for pervasive computing.

A weakness of our architecture is that all the owners of information need to share the same parameters for their HIBE schemes, which could be difficult to achieve. A topic for further investigation is whether we can weaken this assumption without significantly compromising on security. Furthermore, our architecture does not allow secrets required for understanding an obscured proof-of-access description to be forwarded together with a forwarded access right.

In the next chapter, we look at an alternative approach to client-based access control. Namely, we present an encryption-based access-control architecture for pervasive computing. Similar to our solution for obscured proof-of-access descriptions, this architecture exploits hierarchical identity-based encryption. We will also provide a measurement-based evaluation of this approach.

Chapter 7

Encryption-Based Access Control

In this thesis, we have concentrated on client-based access control, where a client needs to prove to a service that the client is authorized to access the requested information. An alternative option is encryption-based access control. In such a scheme, a service provides confidential information to any client, but the service encrypts the information before handing it over to a client. Only clients authorized to access the information have the required decryption key. This approach is attractive in scenarios where there are lots of similar queries to a service since it shields the service from having to run client-specific access control. Instead, the service can encrypt an information item once and use the ciphertext for answering multiple queries for this item. In this chapter, we present an encryption-based access-control architecture for pervasive computing.

7.1 Overview

In Section 1.1, we outlined four challenges for access control in pervasive computing. Let us discuss how we can address these challenges in an encryption-based access-control architecture. First, to keep key management simple, we have a client use only one decryption key for each type of information, even if the same type of information is offered by multiple services, potentially run by different organizations. Second, for complex information, such as a calendar entry, a service offering this information encrypts the information such that a client requires multiple decryption keys, one for each type of revealed information. Third, in order to limit the influence of intruders into a gateway, a gateway can potentially perform its derivation functionality on encrypted information, as outlined in Section 4.9. In this way, a gateway does not need to be given a decryption key. Fourth, when infor-

mation is accessible only if a constraint is satisfied, we have each value of the constraint require a separate decryption key. Constraints can make key management difficult.

In this chapter, we present an encryption-based access-control architecture for pervasive computing that supports constraints and multiple services offering the same information and that keeps key management simple [62]. In particular, our contributions include

- a novel application of hierarchical identity-based encryption in the context of encryption-based access control,
- a prototype implementation and an evaluation of this implementation.

We start by listing the requirements that encryption-based access control needs to fulfill (Section 7.2). We then present our solution based on hierarchical identity-based encryption (Section 7.3). Next, we discuss the scenario where a service returns complex information (Section 7.4). We have implemented our solution in a pervasive computing environment (Section 7.5). Finally, we provide an evaluation and discuss the relative strengths and weaknesses of our example implementation (Section 7.6).

7.2 Requirements

In this section, we discuss the requirements that we require our solution for encryption-based access control to have. The requirements are similar to the requirements we placed on obscured proof-of-access descriptions in client-based access control in Section 6.2.

Indistinguishability. If the information requested by a client is complex, a service cannot inform the client which keys to use for decryption, else there could be an information leak. This observation also calls for indistinguishability, that is, the ciphertext must not reveal any knowledge about the owner of the encrypted information.

Constraints. Each possible value of a constraint must require a separate key for decrypting encrypted information that should be accessible only under the given constraint/value combination. For example, a key that allows decryption on January 1 must not allow decryption on January 2. To make key management simple, we want a key scheme that supports hierarchical constraints. For example, given the decryption key for January, we can derive the key for January 1, January 2,...

Granularity awareness. To simplify key management, the decryption key for coarse-grained information should be derivable from the decryption key for fine-grained information.

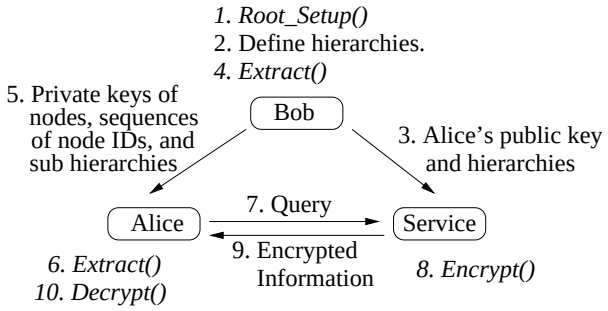


Figure 7.1: Architecture for encryption-based access control. Bob sets up his IBE scheme and hierarchies, informs the service, and grants access to Alice. Alice issues a query.

Asymmetry. Decryption keys should be service independent, that is, if multiple services offer the same information, this information will be decryptable with the same decryption key. Therefore, in a symmetric cryptosystem, a service encrypting information would be able to access the same information offered by some other service. For example, a cellphone service offering Alice's cellphone-based location information would be able to access her Wi-Fi–based location information as offered by another service. We can avoid this problem by using an asymmetric cryptosystem.

We do not require personalization for encryption-based access control since encryption-based access control is client independent by design.

7.3 Encryption-Based Access Control

Similar to obscured proof-of-access descriptions, our architecture for encryption-based access control is based on hierarchical identity-based encryption (HIBE). For the rest of this chapter, we assume familiarity with HIBE, as presented in Section 6.3.1, and our architecture for obscured proof-of-access descriptions, as described in Section 6.3.3.

Figure 7.1 gives an overview of our encryption-based access-control architecture; the architecture is similar to the architecture for obscured proof-of-access descriptions given in Figure 6.1. We now review the changes. For illustration purposes, we assume that the service provides location information.

Setup. There is no need for Bob to personalize his information and constraint hierarchies, since encryption-based access control is not client specific.

Access Control. When queried by Alice for information about Bob (7), the service encrypts the information (8) and returns the encrypted information to Alice (9). Namely, the service splits up the information based on its granularity properties and encrypts each piece separately. For example, the information "CMU Wean Hall 8220" is split up into "CMU", "Wean Hall", and "8220". Then, for each piece, the service locates the node in Bob's information hierarchy that describes the piece and gathers the IDs of all the nodes along the path from the root node to this node. In our example, the ID sequences are ⟨location_fine, medium, coarse⟩, ⟨location_fine, medium⟩, and ⟨location_fine⟩, respectively. Similarly, for each of the constraint hierarchies, the service chooses the leaf node that contains the current value of the constraint and gathers the IDs along the path from the root node. The service then calls $Encrypt()$ with the gathered sequences of node IDs (e.g., $Encrypt(\langle$location_fine, medium, coarse$\rangle, \langle$location_2005, February, 2$\rangle, \langle$location_always, office_hours$\rangle$, "CMU", $Q_0, params)$). Alice decrypts the received ciphertexts by calling $Decrypt()$ with the required tuple of private keys (10) for each ciphertext. She can decrypt a ciphertext only if the encrypted information is of a granularity that she has access to.

Discussion. Our solution fulfills the requirements of indistinguishability (as explained in Section 6.3.3) and asymmetry and supports granularities and multiple, hierarchical constraints. Using the identification string of information or of a constraint directly as its public key drastically simplifies key management.

Security Analysis. As mentioned in Section 7.3, the security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem. The scheme is not secure against collusion. For example, for the hierarchies given in Figure 6.2, assume that Alice has the tuple of private keys for (⟨location_fine⟩, ⟨location_2005⟩, ⟨location_always, office_hours⟩) and that Carol has the tuple for (⟨location_fine⟩, ⟨location_2005, January⟩, ⟨location_always⟩). If Alice and Carol colluded, they could determine the tuple for (⟨location_fine⟩, ⟨location_2005⟩, ⟨location_always⟩). Yao et al. [133] propose a collusion-resistant HIBE scheme. However, the complexity of the $Encrypt()$ and $Decrypt()$ operations in their scheme is $\mathcal{O}(n^m)$, where n is the depth of a hierarchy and m is the number of hierarchies. As we will see in Section 7.6, the complexity of the operations in our scheme is $\mathcal{O}(mn)$.

7.4 Complex Information

If the information offered by a service is complex, decryption of the corresponding ciphertext should require multiple decryption keys, one for each type of revealed information. Let us discuss two options to deal with this scenario.

First, a service determines an ordered set of encryption keys, one for each type of

revealed information, and computes a single encryption key out of them, which it then uses for encrypting the information. (We discuss some approaches for computing this single encryption key, specific to IBE, in Section 7.7.) Similarly, a client picks the corresponding set of decryption keys and computes a single decryption key out of them, which it then uses for decryption. This solution has the advantage that there is always exactly one encryption operation. If there were multiple such operations, a client might be able to learn about their number and deduce confidential information (see below). The approach has the drawback that it does not scale well for the decryption of complex information, where a client does not know the required decryption keys. Namely, the client would have to try all possible combinations of its keys. For example, if a client had n decryption keys, up to $2^n - 1$ decryption attempts would be required.

Second, a service determines a set of encryption keys and performs onion encryption, that is, it randomly removes a key from the set and uses the key to encrypt the information item or the ciphertext resulting from the previous encryption. (Key removal must be random, else information could leak.) This approach has the drawback that it can leak information about the number of applied encryption operations from the final ciphertext. Namely, since the encryption operation is randomized, its application results in a ciphertext longer than the plaintext or ciphertext used as input to the operation. This information leak can be avoided by limiting the maximum number of encryption operations and using a dummy key if the actual number of keys was smaller than this limit. In this way, the final ciphertext will always have the same length. The approach has the advantage that it scales better than the first approach when it comes to locating the required set of keys for decrypting complex information. Given n decryption keys in the client's pool of keys and at most m of them are required for decryption, a client requires at most $\sum_{i=0}^{m-1}(n-i)$ decryption attempts. In our implementation, we choose this approach.

7.5 Prototype Implementation

For encryption-based access control, we re-use the HIBE implementation used for obscuring proof-of access descriptions in client-based access control, as described in Section 6.5.

We also implemented several location services, each exploiting a different approach for locating people. They run either client-based or encryption-based access control. The encryption-based versions always use HIBE. While it is possible to switch to a different asymmetric cryptosystem if, for example, no constraints are used or information is not granularity aware, key management would become difficult.

Due to the reasons outlined in Section 6.5, we use SSL (without client authentication)

for securing the communication between a client and a service.

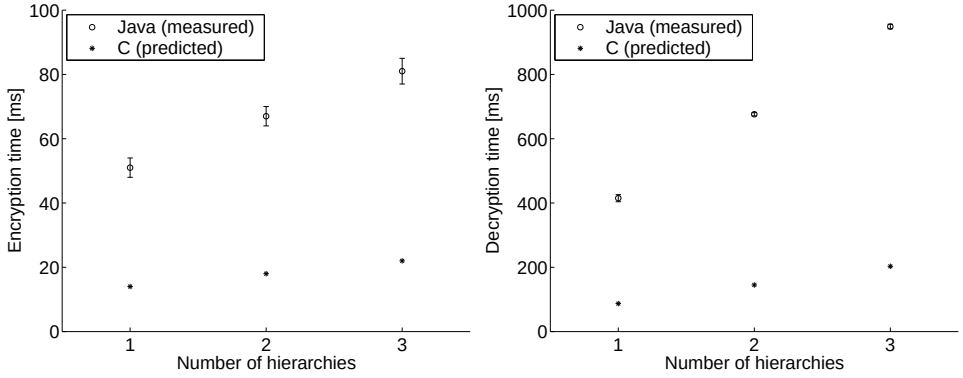
7.6 Evaluation

In our evaluation, we concentrate on encryption-based access control. For obscured proof-of-access descriptions, the overall cost for access control is larger than for encryption-based access control. The performance of the HIBE operations is similar for both cases. However, client-based access control requires two round trips, client authentication, and validation of the proof of access.

We run our experiments on a Pentium IV/2.5 GHz with 1.5 GB of memory, Linux 2.4.20, and Java 1.4.2. An experiment consists of ten runs. We report both the mean and the standard deviation (in parentheses).

We have a client contact a service that provides encrypted people-location information, which is split into three levels of granularity and encrypted using a three-level information hierarchy. There are no constraints. We look only at the case where information about a single individual is provided. In addition, we assume that the client knows which decryption key to use. It takes 1091 ms (42 ms) for the client to retrieve and decrypt the information. Let us examine this cost in more detail. (Detailed results are in Appendix A.3.) For the service, there is a cost of 25 ms (2 ms) for an $Encrypt()$ operation that exploits only the root level of a hierarchy. Our service has to perform three $Encrypt()$ operations. In addition, there is a cost of 14 ms (1 ms) per additional level used in an $Encrypt()$ operation (i.e., $3 * 14$ ms in our experiment). Therefore, the overall cost of encryption is about 117 ms. The overall processing time of the service is 253 ms (31 ms); 46% of the cost is due to encryption. The rest of the cost is caused by fingering a person's desktop computer in order to locate her and by (de)marshalling of the request and the response. For the client, there is a cost of 136 ms (2 ms) per level used in a $Decrypt()$ operation. Our client runs three such operations, operating at 1, 2, or 3 levels. Therefore, the overall decryption cost is about 816 ms or 75% of the overall processing time.

In our second experiment, we investigate the influence of the number of hierarchies on encryption and decryption time. We encrypt and decrypt a random message using a variable number of hierarchies, whereas we exploit all the levels in each hierarchy. Similar to the first experiment, the first hierarchy has three levels. All the additional hierarchies have two levels. As shown in Figure 7.2, the cost increases linearly with the number of hierarchies. This observation is consistent with the characteristics of the $Encrypt()$ and $Decrypt()$ operations (see Appendix A.3). Taking these characteristics into account, if there are m hierarchies having n_i levels $(1 \leq i \leq m)$, the cost of an $Encrypt()$ operation



(a) (b)

Figure 7.2: Performance of encryption/decryption. We encrypt/decrypt a message using a variable number of two-level hierarchies, whereas the first hierarchy has three levels. (The two graphs are differently scaled.)

exploiting all the levels in each hierarchy is 25 ms$+\sum_{i=1}^{m}(n_i-1)*14$ ms. For a $Decrypt()$ operation, the cost is $\sum_{i=1}^{m} n_i * 136$ ms.

The performance numbers heavily depend on the underlying implementation. Our implementation uses Java's standard mathematical package for its cryptographic routines. While we currently do not have a C-based implementation of our HIBE scheme, there is a more optimized, publicly available C-based implementation of the underlying IBE scheme [89]. Since the former scheme exploits the same basic mathematical routines as the latter one, we can predict the performance of a C-based implementation of our HIBE scheme based on this implementation. Figure 7.2 shows our predictions. (More detailed results are in Appendix A.3). Formally, if there are m hierarchies having n_i levels $(1 \leq i \leq m)$, the cost of an $Encrypt()$ operation exploiting all the levels in each hierarchy is 6 ms $+ \sum_{i=1}^{m}(n_i - 1) * 4$ ms. For a $Decrypt()$ operation, the cost is $\sum_{i=1}^{m} n_i * 29$ ms. In summary, the performance of a C-based, more optimized implementation would be at least 3.5 (encryption) or 4.5 (decryption) times better.

We could also improve performance by using a different HIBE scheme. Currently, we are exploiting an extension of Gentry and Silverberg's HIBE scheme [54], which, at the time of the implementation of our architecture, was the most secure and efficient option. Recently, Boneh et al. [25] proposed a HIBE scheme that has constant decryption performance for a single hierarchy, regardless of the number of levels, and that has the

same encryption performance as Gentry and Silverberg's scheme. In addition, a ciphertext in Boneh et al.'s scheme does not leak any information about the number of levels in a hierarchy that were used for encryption.

The presented results allow us to judge the relative benefit, performance-wise, of client-based and encryption-based access control. In our implementation of client-based access control, it takes a service about 2 ms to validate the 1024 bit RSA signature of a SPKI/SDSI certificate. Assuming a single-level information hierarchy and no constraint hierarchies, it takes the service 25 ms to encrypt a piece of information. However, this operation does not need to be executed for every client, the service can re-use an encrypted piece of information to answer requests from multiple clients. Therefore, it pays off for the service to use encryption-based access control if there are more than 12 requests for information during the lifetime of the information. If there are constraints on access rights, this number will become correspondingly larger.

7.7 Related Work

There are several areas of research that are relevant to our work. In particular, let us review research in the areas of access control in a hierarchy, broadcast encryption, secure multicast, and more flexible IBE schemes.

There has been previous work about access control in a hierarchy, where information items are classified into partially ordered security classes depending on their sensitivity and users are assigned to classes depending on their clearance. Each class has an encryption (decryption) key, which is used for encrypting (decrypting) information in the class. Given the encryption (decryption) key for a class, it is possible to derive the encryption (decryption) key for a class of a lower security level. None of the proposed hierarchical architectures fulfills our requirements of asymmetry and easy key management: Akl and Taylor [4], Harn and Yin [59], and Tzeng [122] present symmetric architectures, Sandhu [109] and Zheng et al. [135] propose symmetric architectures exploiting strings for key generation, and Ray et al. [107] discuss an asymmetric approach that does not exploit strings for key generation. Our approach supports only tree-based hierarchies. However, tree-based hierarchies are sufficient for expressing granularity-aware decryption keys and hierarchical constraints on them. Similar to our architecture, Briscoe [31] uses a hierarchy for managing time-based access.

Broadcast encryption [49] is similar to encryption-based access control, as explored in this chapter. In broadcast encryption, a service encrypts a message for a subset of clients who are listening on a broadcast channel. A client in this subset can decrypt the broadcast,

whereas (colluding) clients outside of the subset cannot. Since information is broadcast to clients with potentially limited storage capabilities, research in this area (e.g., Boneh et al. [27]) is mainly concerned with limiting the size of ciphertexts and of private keys. In our scenario, there is no broadcasting of information. We assume that clients have sufficient storage capabilities. Alternatively, clients can offload the storage of decryption keys and potentially the decryption of ciphertexts to more powerful proxies. Broadcast encryption assumes that a client either has or does not have a decryption key; there are no constraints on the validity of a decryption key.

In broadcast encryption, it is the responsibility of a service to deal with joining or leaving clients. In secure multicast [131, 124], clients need to re-key their decryption keys after a join or leave event, that is, clients need to keep state. Research in this area is mainly concerned with efficient re-keying (e.g., Sherman and McGrew [112]). In our approach, the issuer of a decryption key decides about the lifetime of the key at the time when the key is issued and embeds the lifetime in the key; there is no need for a client to update the key while using the key.

The Boneh and Franklin IBE scheme [26] supports scenarios where a client needs to have a particular decryption key in order to decrypt a ciphertext. Several research projects have studied ways to make this scheme more flexible. Chen et al.'s scheme [40] supports conjunction (i.e., the client needs to have multiple decryption keys) and restricted disjunction (i.e., the client needs to have one out of two decryption keys). The decryption keys can be handed out by different authorities. Smart's scheme [113] supports scenarios where the required decryption keys can be expressed in the form of a "conjunctive-DNF" (i.e., $\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i}(\bigwedge_{k=1}^{m_{i,j}} s_{i,j,k}))$, where $s_{i,j,k}$ denotes the statement that "the client has the decryption key corresponding to the identity $ID_{i,j,k}$"). The scheme supports only a single issuer of decryption keys. Bagga and Molva [11] extend Smart's scheme to allow for multiple issuers of decryption keys. The three schemes assume that the client knows about the required decryption keys, which does not always hold in our scenario. We require only conjunction, which we achieve based on onion encryption. As mentioned in Section 7.4, in our scheme, a client needs to perform at most $\sum_{i=0}^{m-1}(n-i)$ decryption attempts, where n corresponds to the number of decryption keys issued to the client and m is the maximum number of keys required for decryption. The corresponding number for the alternative schemes presented in this section is $2^n - 1$.

7.8 Summary

We presented an architecture for encryption-based access control in pervasive computing, based on hierarchical identity-based encryption. Our solution supports information of different granularity levels and makes it possible to constrain a client's ability to decrypt a ciphertext.

The evaluation of our architecture showed that identity-based encryption is expensive. However, the overhead can be significantly lowered using a more optimized implementation. Furthermore, our architecture gives us the convenience of being able to use the name of information or of a constraint as public key.

Chapter 8

Conclusions and Future Work

This dissertation set out to address challenges in access control to information in pervasive computing environments. In particular, these challenges were:

Diversity in service administration. There will be many services, potentially offering the same information, run by different organizations, even in a single social environment.

Complex information. There will be complex information, revealing multiple types of information, which could result in information leaks.

Derivation of information. Services that derive information will become attractive targets for attackers.

Confidential context-sensitive constraints. Confidential information will be used for constraining access, which could result in information leaks.

In this final chapter, we summarize our contributions toward solving these challenges. Moreover, we reflect on their application and their limitations. Finally, we also describe some directions for future work.

8.1 Contributions

The main contributions of this thesis are:

Client-based access-control architecture. We presented a distributed access-control architecture for pervasive computing, in which clients need to prove to a service that they are authorized to access the requested information (Chapter 2). We also defined an information-representation scheme, and we argued that such a scheme should be independent of the service providing the represented information. The architecture and the representation scheme proved flexible enough to support the other contributions made in this thesis.

Information relationships. We proposed exploiting the semantics of information for access control (Chapter 3). In particular, we capture the semantics of complex information with information relationships. Furthermore, information relationships let individuals bundle information that has identical access requirements and reduce the number of access rights that individuals need to issue. We also argued that, in pervasive computing, an owner of complex information should not be prevented from granting explicit access rights to this information. We showed that information relationships are feasible in practice based on an analytical evaluation and on a measurement-based evaluation of an implementation within the client-based access-control architecture.

Derivation-constrained access control. We proposed the formalization of derivation properties between information and exploiting these properties for access control (Chapter 4). Namely, we require services that derive information to prove that they are asking for information in order to answer a request for derived information. Based on these two concepts, we reduce the influence of an intruder into such a service. We outlined a trade-off between the flexibility in defining statements like access rights or derivation properties and the security properties of our algorithm. Finally, we showed that the proposed concepts are practical based on a measurement-based implementation within the client-based access-control architecture.

Context-sensitive access control. We studied information leaks caused by context-sensitive constraints in multiple access-control approaches (Chapter 5). We proposed the usage of access-rights graphs for the identification of conflicting constraints and for the resolution of constraints. Moreover, we introduced hidden constraints and showed, based on two sample implementations of this concept, how we can keep constraints secret from a service. We argued that keeping access rights confidential can avoid some information leaks and that constraints should be limited in their nature to make it easier to check that constraints do not result in information leaks. Our implementation within the client-based access-control architecture suggests that constraints can be efficiently dealt with in practice.

Obscured proof-of-access descriptions. We developed an approach for obscuring the description of a required proof of access, as returned by a service to a client, based on hierarchical identity-based encryption (Chapter 6). Our approach supports both different granularities of information and constraints on access rights to information and has many advantages when compared with approaches based on traditional asymmetric cryptosystems. Whereas our sample implementation is costly in terms of performance, we outlined several options to improve this performance.

Encryption-based access-control architecture. As an alternative to client-based access control, we presented an encryption-based access-control architecture for pervasive computing based on hierarchical identity-based encryption. Our architecture supports both different granularities of information and constraints on access rights to information. Again, our sample implementation is costly in terms of performance, but there is the potential for improvements.

8.2 Reflections

In this section, we apply the contributions presented in this thesis to two example scenarios. The scenarios demonstrate how the contributions work in combination. Moreover, the scenarios help outline some limitations of our contributions. The scenarios involve an office and a home environment.

8.2.1 Office Scenario

The first scenario corresponds to the office scenario that we introduced in Section 1.1 and that we have used throughout the thesis. The scenario illustrates the application of client-based access control, combination-based information relationships, and confidential context-sensitive constraints.

In the scenario, Carol uses a calendar application provided by her employer. Since the application supports client-specific access right (i.e., different clients can have access rights with different kinds of constraints), it employs client-based access control, instead of encryption-based access control. Furthermore, there is no gateway in this scenario, so there is no need for derivation-constrained access control.

Carol wants to respect the privacy of people that she is meeting with. Therefore, for a meeting with Bob, she defines a combination-based information relationship between the corresponding calendar entry and Carol and Bob's location and activity information. (In

the implementation, Carol's calendar application defines this relationship on Carol's behalf when Carol sets up a meeting.) When Dave requests access to Carol's calendar entry, he is informed by the calendar service of the required proof of access. In particular, the service tells Dave that he needs to prove that he can access Carol and Bob's location and activity information. However, to avoid an information leak, the service cannot reveal Bob's identity to Dave. Therefore, the service creates an obscured proof-of-access description asking for Bob's location. (There is no need to create an obscured proof-of access description asking for Bob's activity. Since the composition of calendar information is well known, Dave can infer that he also needs to submit an access right to Bob's activity information after understanding the obscured description asking for Bob's location information.) In addition, the service directly asks for access rights to Carol's location and activity information. Dave searches his secrets to understand the obscured proof-of-access description. He will succeed if he is in the possession of an access right to Bob's location information. Next, he submits access rights to Carol and Bob's location and activity information to the calendar service. In particular, he submits four proofs showing that Dave can speak for Carol and Bob regarding their location and activity information. The calendar service combines these proofs with the combination-based relationship and verifies the combined proof. If the verification succeeds, the service will return Carol's calendar entry to Dave.

Carol wants her executive assistant, Alice, to have access to her calendar entry, regardless of the access rights defined by people that Carol is meeting with. Therefore, Carol grants Alice an access right to the entry. The access right is constrained to Carol being in Pittsburgh and to Carol not being on vacation. Carol uses hidden constraints when defining these constraints to make it easier for Alice to verify whether she can send a proof of access to the calendar service. When Alice wants to access Carol's calendar entry, Alice builds an access-rights graph, which tells her that she first needs to get assurances for Carol's location and activity from the corresponding services. Assuming that Carol has granted Alice access rights to this information, Alice can assemble them in proofs of access and retrieve the assurances. Next, she assembles the assurances and her access right to the calendar information in a proof of access. Before sending this proof to the calendar service, Alice must ensure that Carol's location and activity information do not leak to the issuer of the access right. This check is straightforward, because the constraints in the access right are about the issuer of the access right, and the issuer has access to this constraint information. The constraints in the access right to the calendar information are hidden, so Alice does not have to verify whether the calendar service has access to Carol's location and activity information. Therefore, Alice can send her proof of access to the service, which will verify the proof and return the requested information.

Let us now discuss limitation of our contributions in the context of the given scenario:

- If Carol meets with a person whose location or activity information Alice is not authorized to access, Alice will still be able to learn this information from the calendar entry. This approach corresponds to already existing, widely accepted social behavior, that is, granting an executive assistant access to her boss' calendar. Moreover, the usefulness of a service could be drastically reduced otherwise.

- The secrets required for understanding obscured proof-of-access descriptions are not forwardable. If Dave has access to Carol's calendar entry based on the combination-based relationship, he can forward the access rights that grant him access to a third entity. However, he cannot forward the secrets that allow him to understand an obscured proof-of-access description generated by the calendar service. Therefore, the third entity would have to submit all potentially relevant access rights to the calendar service.

- Alice retrieves assurances for Carol's location and activity information before accessing Carol's calendar entry. It is possible that Carol's location or activity changes between the time the assurances are issued and the time the calendar information is accessed. Timestamped queries can avoid this staleness problem. We do not consider staleness of context-sensitive information to be a severe issue for pervasive computing and let a service that provides context-sensitive information indicate for how long it expects an assurance to remain valid.

- There are attacks that we do not address in the thesis and that might allow an attacker to learn (some) information about Carol's calendar entry, without having an access right to this information. For example, the attacker could walk by Carol's office and learn Bob's current location and activity.

8.2.2 Home Scenario

Let us now discuss a home scenario. Its main difference, when compared with the first scenario, is the existence of a service acting as a gateway. Namely, the scenario illustrates the application of derivation-constrained access control and bundling-based information relationships.

Dave has a sensor network in his home. Among other information, this network provides location information about Dave and his family. In particular, a location service in Dave's house gathers sensor information and uses this information for deriving people's location information. Dave would like to limit the impact of intruders into this service. The information returned by the sensors is of a raw nature. Therefore, having a sensor return

encrypted information and having the location service compute on encrypted information is not an option. Instead, Dave employs derivation-constrained access control and grants the location service only derivation-constrained access to the raw sensor information.

Furthermore, whereas Dave might want friends of the family to have access to the family's location information, he does not want them to have access to the raw sensor information. Therefore, he grants the friends only access rights to the location information. (Dave could constrain these access rights in a similar way as outlined in the first scenario.) In this setup, the location service can access the raw sensor information only if there is an authorized request from one of the family's friends for location information.

Dave employs many different types of sensors (thermostat, light, weight,..), and there are many instances of each sensor type. Dave wants each family member to have access to the information returned by all sensors. In addition, a house sitter should (temporarily) have the same access. Instead of manually defining access rights for each sensor and individual, Dave bundles the information provided by the sensors in a single type of information and grants individuals access to the bundle. This strategy decreases the number of statements that Dave needs to issue. To grant a house sitter access, Dave issues a single access right to the entire information bundle, where the access right has a limited lifetime. When incorporating a new sensor, Dave defines a bundling-based relationship that bundles the information provided by the sensor in the single type of information.

Let us now discuss limitation of our contributions in the context of the given scenario:

- An attacker into the location service still gets to see the raw sensor information returned to the location service when a friend of the family sends an authorized request for location information to the location service.

- For some sensors, the location service needs to continuously issue queries to a sensor, not just upon receiving a query from a client (e.g., a door sensor registering people entering/leaving a room). Derivation-constrained access control is not applicable here, and the location service needs unconstrained access rights. Therefore, an attacker into the service will be able to issue authorized queries for some sensor information.

- While bundling-based relationships reduce the number of decisions that an individual needs to make, they increase the set of options that are available to Dave and that we expect Dave to be familiar with. We expect the concept to be familiar since it is also used for other applications. However, only a wide-scale deployment of our mechanisms can validate this expectation.

8.3 Directions for Future Work

This dissertation opens up new interesting directions for future research. In this section, we outline some directions.

8.3.1 Remote Discovery of Certificates

In this thesis, we assumed that a client building a proof of access has the required certificates expressing access rights or information relationships locally available. This assumption might not hold in practice. The assumption holds if there are no information relationships and if all access rights are issued before the client tries to access information. In such a scenario, an issuer of an access right can give the subject both the access right and any access rights that grant access to the issuer. For example, Tamassia et al. [120] explore this scenario. However, if (some) access rights are issued only upon a request, the assumption breaks and a client needs to perform remote discovery of access rights before issuing the request. Bauer et al. [15] examine this scenario. Alternatively, a service can perform this search, as studied by Minami and Kotz [93]. The assumption also breaks if access rights are more flexible (e.g., access is granted to an entire group of people, as explored by Li et al. [88]) or if there are information relationships. Here, the problem is that group membership certificates or information relationships could be defined after an access right that exploits them is defined, so the issuer of the access right cannot hand them over to the subject(s) of the access right when issuing this access right. Furthermore, credentials may be confidential, which can complicate a client's search. For example, a company might not be willing to publicly release membership certificates for all its employees. Similarly, access rights, information relationships, and membership certificates can contain sensitive information, which a client might be reluctant to reveal to a service in a proof of access.

8.3.2 Semantics for Access-Control Logic

We have not proved soundness of our extensions to Lampson et al.'s theory of authentication. Abadi et al. [1] prove soundness of this logic based on a semantic model of the logic. Similarly, Howell and Kotz [68] provide a semantic model for their extensions to the logic (i.e., the restricted speaks-for relationship). It is possible to give our extensions a semantics in this model. However, this model is rather complex. An alternative approach is to define our axioms in a higher-order logic, which is known to be sound. In Section 3.9, we explored this approach by adding information relationships to Bauer's proof-carrying

authorization framework [13]. We can incorporate our other contributions into this framework along the same lines.

8.3.3 Semantic-Web Concepts

In the context of the Semantic Web [17], many ontologies for representing knowledge and tools for reasoning about this knowledge have been developed. An interesting question is whether we can enhance our access-control architecture with these ontologies and tools? For example, we could exploit them for defining and reasoning about information relationships. Potentially, these relationships could be more complex than the ones introduced in Chapter 3. Along the same lines, we can investigate the benefits of combining different approaches for running access control in pervasive computing. For instance, there are some research projects that already make usage of ontologies and Semantic Web tools for access control [38, 51]. However, these projects assume that an owner of information manages the access policy for this information and do not support distributed credentials, as we do in our access-control architecture.

8.3.4 Uncertainty in Access Control

Access rights can be constrained to, for example, someone's location. However, due to the limited accuracy of technologies for locating people, often there is uncertainty about a person's location. Access control must be able to deal with such uncertainty. Some interesting questions are: When an individual grants another individual access to personal information under the constraint that this person is at a particular location, is the individual willing to deal with (some) uncertainty? How do we handle multiple services that provide (conflicting) location information?

8.4 Summary

Apart from the technical contributions made in this thesis, an informal contribution is to make the public, in particular, researchers in the area of pervasive computing, aware of some information leaks that arise when existing access-control approaches are naïvely applied to these new computing environments. These leaks will violate people's privacy. Unfortunately, as found in a survey [84], researchers in pervasive computing are not necessarily aware of or worried about these privacy violations. Lahlou et al. [80] summarize this survey as follows:

> Privacy was either an abstract problem; not a problem yet (they are "only prototypes"); not a problem at all (firewalls and cryptography would take care of it); not *their* problem (but one for politicians, lawmakers, or, more vaguely, society); or simply not part of the project deliverables.

The Internet is the primary example of a research project that largely ignored security issues and that (unexpectedly?) grew into what it is today. Let us not repeat this mistake for pervasive computing.

Appendix A

Hierarchical Identity-Based Encryption

A.1 Operations

In this section, we describe the HIBE operations introduced in Section 6.3 in detail. Our operations are based on the operations proposed by Gentry and Silverberg [54], we extend these operations to support multiple hierarchies. (We merge the $Lower\_Level\_Setup()$ and $Extract()$ operations.) We also assume that there is a global set of public parameters. Gentry and Silverberg present two encryption schemes, a semantically secure one and a scheme secure against adaptive chosen ciphertext attacks in the random oracle model. For presentation purposes, we base our discussion on the semantically secure scheme. It is straightforward to generalize our scheme to a scheme secure against chosen ciphertext attacks.

There is a set of public parameters $params = (\mathbb{G}_1, \mathbb{G}_2, q, \hat{e}, P_0, H_1, H_2)$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of some prime order q, $\hat{e}$ is an admissible paring: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, P_0 is an arbitrary generator of $\mathbb{G}_1$, and H_1 and H_2 are cryptographic hash functions $\{0,1\}^* \rightarrow \mathbb{G}_1$ and $\mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n, respectively. One of the properties of an admissible pairing is bilinearity: $\hat{e}(aQ, bR) = \hat{e}(Q, R)^{ab}$ for all $Q, R \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.

- $Root\_Setup(params) \rightarrow Q_0$:
  Choose a random master secret, $s_0 \in \mathbb{Z}/q\mathbb{Z}$, and return $Q_0 = s_0 P_0$.
- $Extract(\langle ID_{i,1}, \ldots, ID_{i,t_i} \rangle, S_{i,t_i-1}, params) \rightarrow S_{i,t_i}, \langle Q_{i,1}, \ldots, Q_{i,t_i-1} \rangle$ with $t_i \geq 1$:

  1. If $t_i > 1$, pick random, secret $s_{i,t_i-1} \in \mathbb{Z}/q\mathbb{Z}$. Otherwise $s_{i,0} = s_0$.

2. Compute $P_{i,t_i} = H_1(ID_{i,1}, \ldots, ID_{i,t_i}) \in \mathbb{G}_1$.
3. Compute secret point $S_{i,t_i} = S_{i,t_i-1} + s_{i,t_i-1}P_{i,t_i} = \sum_{j=1}^{t_i} s_{i,j-1}P_{i,j}$. ($S_{i,0}$ is the identity element of $\mathbb{G}_1$.)
4. Compute (non-secret) $Q_{i,j} = s_{i,j}P_0$ for $1 \leq j \leq t_i - 1$.

- $Encrypt(\langle ID_{1,1}, \ldots, ID_{1,t_1}\rangle, \ldots, \langle ID_{h,1}, \ldots, ID_{h,t_h}\rangle, M, Q_0, params) \rightarrow C$:

  1. Compute $P_{i,j} = H_1(ID_{i,1}, \ldots, ID_{i,j}) \in \mathbb{G}_1$ for $1 \leq j \leq t_i$ and $1 \leq i \leq h$.
  2. Choose a random $r \in \mathbb{Z}/q\mathbb{Z}$.
  3. Set the ciphertext to be:

  $$C = [rP_0, \langle rP_{1,2}, \ldots, rP_{1,t_1}\rangle, \ldots \langle rP_{h,2}, \ldots, rP_{h,t_h}\rangle, M \oplus H_2(g^r)]$$

  where

  $$g = \prod_{i=1}^{h} \hat{e}(Q_0, P_{i,1}) \in \mathbb{G}_2.$$

- $Decrypt(\langle S_{1,t_1}, \ldots, S_{h,t_h}\rangle, \langle Q_{1,1}, \ldots, Q_{1,t_1-1}\rangle, \ldots, \langle Q_{h,1}, \ldots, Q_{h,t_h-1}\rangle, C, params) \rightarrow M$:

  Let $C = [U_0, \langle U_{1,2}, \ldots, U_{1,t_1}\rangle, \ldots, \langle U_{h,2}, \ldots, U_{h,t_h}\rangle, V]$ be the ciphertext encrypted using the sequences of IDs $\langle ID_{1,1}, \ldots, ID_{1,t_1}\rangle, \ldots, \langle ID_{h,1}, \ldots, ID_{h,t_h}\rangle$. To decrypt C, compute

  $$V \oplus H_2\left(\frac{\prod_{i=1}^{h} \hat{e}(U_0, S_{i,t_i})}{\prod_{i=1}^{h}\prod_{j=2}^{t_i} \hat{e}(Q_{i,j-1}, U_{i,j})}\right) = M.$$

The security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem: Given a randomly chosen $P \in \mathbb{G}_1$, as well as aP, bP, and cP (for unknown randomly chosen $a, b, c \in \mathbb{Z}/q\mathbb{Z}$), compute $\hat{e}(P, P)^{abc}$. For proving security of our scheme in the random oracle model, we have to modify Gentry and Silverberg's proof [54, Appendix A.3]. These modifications are straightforward, they exploit the same ideas that we have exploited for extending Gentry and Silverberg's scheme. We give only an outline of the modifications here: There needs to be a separate H_1^{list} for each hierarchy, and the

challenge operation needs to take the additional hierarchies into account. For the latter modification, we rely on the symmetry of $\hat{e}$. In addition, the game between the challenger and the adversary needs to be extended such that the adversary can issue private key extraction queries for multiple PKGs. In particular, the adversary can issue extraction queries for any $(Q_0, \langle ID_{1,1}, \ldots, ID_{1,t_1} \rangle, \ldots \langle ID_{h,1}, \ldots, ID_{h,t_h} \rangle)$, as long as either Q_0 is different from the Q_0 on which the adversary is challenged or none of the ID sequences is identical to the corresponding sequence (or a prefix of this sequence) in the set of sequences on which the adversary is challenged.

The alternative solution for client-based access control suggested in Section 6.3.3 relies on the hardness of the Decision Bilinear Diffie-Hellman problem: Given a randomly chosen $P \in \mathbb{G}_1$, as well as aP, bP, cP, and r (for some $a, b, c, r \in \mathbb{Z}/q\mathbb{Z}$), return true if $r = \hat{e}(P, P)^{abc}$.

A.2 Optimizations

In this section, we discuss a few optimizations that we applied in our implementation of the HIBE operations outlined in Appendix A.1.

$Encrypt()$ is a time-critical operation since it is run by a service. To speed up this operation, a service can precompute the $P_{i,j}$ in step 1 and the pairings in step 3. In addition, the service can precompute sliding windows for multiplications on $P_{i,j}$ and use them for the computation of $rP_{i,j}$.

A.3 Evaluation

Our Java-based implementation, which is based on a C implementation [56], exploits Tate pairings over super-singular elliptic curves. We use a 160 bit prime for q and a 512 bit prime for p, where $\mathbb{G}_1$ is an order-q subgroup of $E(\mathbb{F}_p)$ and $\mathbb{G}_2$ is an order-q subgroup of $\mathbb{F}_{p^2}$. For the configuration given in Section 7.6, the performance of the expensive cryptographic operations is given in Table A.1. For the C-based implementation [89], we use the same set of parameters and configuration as for the Java-based version (gcc 3.2.3 instead of Java 1.4.2).

Operation	Java		C	
	μ	(σ)	μ	(σ)
$rP_{i,j}$	14	(1)	4	(0)
Exponentiation	11	(1)	2	(0)
Pairing	136	(2)	29	(0)

Table A.1: Mean and standard deviation of elapsed time for expensive cryptographic operations in our Java-based implementation and in MIRACL's C-based implementation [89] [ms].

Bibliography

[1] M. Abadi, M. Burrows, and B. Lampson. A Calculus for Access Control in Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15 (4):706–734, October 1993. 2.6.2, 4.4, 8.3.2

[2] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In *Proceedings of CRYPTO 2005*, pages 205–222, August 2005. 6.3.3

[3] N. R. Adam and J. C. Wortmann. Security-Control Methods for Statistical Databases: A Comparative Study. *ACM Computing Surveys (CSUR)*, 21(4):515–556, December 1989. 5.4, 5.10

[4] S. G. Akl and P. D. Taylor. Cryptographic Solution to a Problem of Access Control in a Hierarchy. *ACM Transactions on Computer Systems*, 1(3):293–248, 1983. 7.7

[5] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. D. Mickunas. Cerberus: A Context-Aware Security Scheme for Smart Spaces. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 489–496, March 2003. 1.2.1, 1.2.2, 1.2.4, 2.3.1, 2.8.2, 2.8.3, 5.10

[6] Teen Arrive Alive. `http://www.teenarrivealive.com`. 1

[7] A. W. Appel and E. W. Felten. Proof-Carrying Authentication. In *Proceedings of 6th ACM Conference on Computer and Communications Security*, November 1999. 2.3.1, 2.3.3, 2.8.3

[8] P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-P3P Privacy Policies and Privacy Authorization. In *Proceedings of Workshop on Privacy in the Electronic Society (WPES)*, pages 103–109, November 2002. 2.1, 2.2.2

[9] V. Atluri and A. Gal. An Authorization Model for Temporal and Derived Data: Securing Information Portals. *ACM Transactions on Information and System Security*, 5(1):62–94, February 2002. 3.10

[10] J. Bacon, K. Moody, and W. Yao. A Model of OASIS Role-Based Access Control and its Support for Active Security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, November 2002. 5.10

[11] W. Bagga and R. Molva. Policy-Based Cryptography and Applications. In *Proceedings of 9th International Conference of Financial Cryptography and Data Security (FC'05)*, 2005. 7.7

[12] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H.-C. Wong. Secret Handshakes from Pairing-Based Key Agreements. In *Proceedings of 2003 IEEE Symposium on Security and Privacy*, pages 180–196, May 2003. 6.4.1, 6.6

[13] L. Bauer. *Access Control for the Web via Proof-Carrying Authorization*. PhD thesis, Princeton University, November 2003. 3.1, 3.9, 8.3.2

[14] L. Bauer. Personal Communication, June 2005. 3.9.2

[15] L. Bauer, S. Garriss, and M. K. Reiter. Distributed Proving in Access-Control Systems. In *Proceedings of 2005 IEEE Symposium on Security and Privacy*, pages 81–95, May 2005. 8.3.1

[16] L. Bauer, M. A. Schneider, and E. W. Felten. A General and Flexible Access-Control System for the Web. In *Proceedings of 11th Usenix Security Symposium*, pages 93–108, August 2002. 1.3, 2.2.1, 2.3.3, 2.8.2, 2.8.3, 3.6.3, 3.9.2, 3.9.3, 3.9.5, 4.7

[17] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2002. 1.2.2, 8.3.3

[18] E. Bertino, C. Bettini, and P. Samarati. A Temporal Authorization Model. In *Proceedings of 2nd ACM Conference on Computer and Communications Security (CCS 1994)*, pages 126–135, November 1994. 4.9

[19] E. Bertino, F. Origgi, and P. Samarati. A New Authorization Model for Object-Oriented Databases. *Database Security, VIII (A-60)*, pages 199–222, 1994. 3.10

[20] M. Bishop. *Computer Security: Art and Science*. Addison Wesley, November 2002. ISBN 0-201-44099-7. 2.1

[21] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of 17th IEEE Symposium on Security and Privacy*, pages 164–173, 1996. 2.2.1, 2.8.4

[22] M. Blaze, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, September 1999. 2.2.1, 2.8.4, 3.10

[23] BodyMedia. `http://www.bodymedia.com`. 1

[24] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In *Proceedings of EUROCRYPT 2004*, pages 223–238, May 2004. 6.5

[25] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *Proceedings of Eurocrypt 2005*, May 2005. 6.5, 7.6

[26] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. Extended Abstract in Proceedings of Crypto 2001, pp. 213-229, 2001. 6.3.1, 6.3.3, 6.4.1, 6.5, 6.6, 7.7

[27] D. Boneh, C. Gentry, and Waters. B. Collusion Resistant Broadcast Encryption With Short Ciphertexts and Private Keys. Cryptology ePrint Archive: Report 2005/018, January 2005. 7.7

[28] N. Borisov and E. Brewer. Active Certificates: A Framework for Delegation. In *Proceedings of Network and Distributed System Security Symposium (NDSS '02)*, February 2002. 2.8.4

[29] R. Bradshaw, J. Holt, and K. E. Seamons. Concealing Complex Policies with Hidden Credentials. In *Proceedings of 11th ACM conference on Computer and Communications Security (CCS 2004)*, pages 146–157, October 2004. 6.3.4, 6.6

[30] S Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000. ISBN 0-262-02491-8. 6.6

[31] B. Briscoe. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In *Proceedings of First International Workshop on Networked Group Communication*, pages 301–320, November 1999. 7.7

[32] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison Wesley Professional, November 1994. ISBN 0-201-59375-0. 2.1, 5.4, 5.10

[33] C. Castelluccia, S. Jarecki, and G. Tsudik. Secret Handshakes from CA-Oblivious Encryption. In *Proceedings of Asiacrypt 2004*, December 2004. 6.4.1, 6.6

[34] A. Cavoukian. Building in Privacy from the Bottom up: How to Preserve Privacy in a Security-Centric World. Talk given at Carnegie Mellon University, November 2004. 2.1

[35] D. Chaum. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985. 6.6

[36] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems*, 18(3): 197–207, May 2003. 2.8.3

[37] H. Chen, T. Finin, and A. Joshi. A Pervasive Computing Ontology for User Privacy Protection in the Context Broker Architecture. Technical Report TR-CS-04-08, Department of Computer Science & Electrical Engineering, University of Maryland, 2004. 2.8.3

[38] H. Chen, T. Finin, and A. Joshi. Semantic Web in the Context Broker Architecture. In *Proceedings of 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)*, pages 277–286, March 2004. 1.2.1, 1.2.2, 1.2.4, 2.3.1, 3.10, 5.10, 6.4.1, 8.3.3

[39] H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *Proceedings of First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004)*, August 2004. 2.5.1, 2.8.2

[40] L. Chen, K. Harrison, D. Soldera, and N. Smart. Applications of Multiple Trust Authorities in Pairing Based Cryptosystems. In *Proceedings of International Conference on Infrastructure Security (InfraSec 2002)*, pages 260–275, October 2002. 7.7

[41] C. Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Proceedings of 8th IMA International Conference on Cryptography and Coding*, pages 360–363, December 2001. 6.5

[42] M. J. Covington, P. Fogla, Z. Zhan, and M. Ahamad. A Context-Aware Security Architecture for Emerging Applications. In *Proceedings of 18th Annual Computer*

Security Applications Conference (ACSAC 2002), December 2002. 1.2.1, 1.2.2, 2.3.1, 2.8.2, 2.8.3, 5.10

[43] M. J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd. Securing Context-Aware Applications Using Environment Roles. In *Proceedings of 6th ACM Symposium on Access Control Models and Technologies (SACMAT '01)*, pages 10–20, May 2001. 5.10

[44] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation, April 2002. 2.8.3

[45] M. Dean and G. Schreiber. OWL Web Ontology Language Reference. W3C Recommendation, February 2004. 2.5.1, 2.8.3

[46] D. E. Denning. *Cryptography and Data Security*. Addison Wesley, June 1982. ISBN 0-201-10150-5. 5.4, 5.10

[47] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, September 1999. 2.2.1, 2.4.2, 2.5.1, 2.5.3, 2.8.4, 3.10, 5.8.3, 6.3.1

[48] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. SPKI Examples. Internet Draft, March 1998. `http://theworld.com/~cme/examples.txt`. 6.5

[49] A. Fiat and M. Naor. Broadcast Encryption. In *Proceedings of Crypto '93*, pages 480–491, August 1993. 7.7

[50] K. Frikken, M. Atallah, and J. Li. Hidden Access Control Policies with Hidden Credentials. In *Proceedings of Workshop on Privacy in the Electronic Society (WPES)*, pages 27–28, October 2004. 6.6

[51] F. Gandon and N. Sadeh. A Semantic eWallet to Reconcile Privacy and Context Awareness. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, October 2003. 1.2.1, 1.2.2, 1.2.4, 2.5.1, 2.8.2, 2.8.3, 3.10, 5.10, 6.4.1, 8.3.3

[52] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, April-June 2002. 2.6.4

[53] M. Gasser and E. McDermott. An Architecture for Practical Delegation in a Distributed System. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 20–30, May 1990. 4.2, 4.5.2, 4.9

[54] C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptography. In *Proceedings of Asiacrypt 2002*, pages 548–566, December 2002. 6.3.1, 6.3.2, 6.3.3, 7.6, A.1, A.1

[55] P. G. Griffiths and B. Wade. An Authorization Mechanism for a Relational Database System. *ACM Transactions on Database Systems*, 1(3):242–255, 1976. 1.1

[56] Stanford Applied Crypto Group. IBE Secure E-mail. `http://crypto.stanford.edu/ibe`. 6.5, A.3

[57] V. Gupta, M. Millard, S. Fung, F. Zhu, N. Gura, H. Eberle, and S. Chang. Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet. In *Proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, pages 247–256, March 2005. 2.3.4

[58] N. Hardy. The Confused Deputy (or why capabilities might have been invented). *Operating Systems Review*, 22(4):36–38, October 1988. 4.6

[59] L. Harn and H. Y. Lin. A Cryptographic Key Generation Scheme for Multi-level Data Security. *Computer & Security*, 9(6):539–546, 1990. 7.7

[60] U. Hengartner and P. Steenkiste. Access Control to Information in Pervasive Computing Environments. In *Proceedings of 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, pages 157–162, May 2003. 1.1

[61] U. Hengartner and P. Steenkiste. Implementing Access Control to People Location Information. In *Proceedings of 9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 11–20, June 2004. 2.4, 2.4.4

[62] U. Hengartner and P.. Steenkiste. Exploiting Hierarchical Identity-Based Encryption for Access Control to Pervasive Computing Information. To appear in Proc. of First IEEE/CreateNet International Conference on Security and Privacy for Emerging Areas in Communication Networks (IEEE/CreateNet SecureComm 2005), September 2005. 6.1, 7.1

[63] U. Hengartner and P. Steenkiste. Exploiting Information Relationships for Access Control. In *Proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, pages 269–278, March 2005. 3

[64] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):57–66, August 2001. 2.8.1

[65] J. Holt, R. W. Bradshaw, K. E. Seamons, and H. Orman. Hidden Credentials. In *Proceedings of 2nd ACM Workshop on Privacy in the Electronic Society*, October 2003. 6.3.3, 6.6

[66] J. I. Hong and J. A. Landay. An Architecture for Privacy-Sensitive Ubiquitous Computing. In *Proceedings of Second International Conference on Mobile Systems, Applications, and Services (Mobisys 2004)*, pages 177–189, June 2004. 1.2.1, 2.1, 2.5.1, 2.8.2, 3.10

[67] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988. 1.1, 1.1.1, 2.2.1

[68] J. Howell and D. Kotz. A Formal Semantics for SPKI. In *Proceedings of 6th European Symposium on Research in Computer Security (ESORICS 2000)*, pages 140–158, October 2000. 2.5.2, 8.3.2

[69] J. Howell and D. Kotz. End-to-end authorization. In *Proceedings of 4th Symposium on Operating System Design & Implementation (OSDI 2000)*, pages 151–164, October 2000. 1.3, 2.2.1, 2.3.3, 2.3.4, 2.4.2, 2.6.3, 2.8.2, 2.8.4, 4.2, 4.5.1, 4.9

[70] Y.-C. Hu, M. Jakobsson, and A. Perrig. Efficient Constructions for One-way Hash Chains. In *Proceedings of Applied Cryptography and Network Security (ACNS 2005)*, June 2005. 5.8.2

[71] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmaninan. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems*, 26 (2):214–260, June 2001. 3.10

[72] T. Jim. SD3: A Trust Management System with Certified Evaluation. In *Proceedings of 2001 IEEE Symposium on Security and Privacy*, pages 106–115, May 2001. 2.8.3, 2.8.4

[73] G. Judd and P. Steenkiste. Providing Contextual Information to Ubiquitous Computing Applications. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 133–142, March 2003. 2.6.4

[74] L. Kagal, T. Finin, and A. Joshi. A Policy Language for a Pervasive Computing Environment. In *Proceedings of 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–76, June 2004. 2.8.3

[75] T. Kagal, L. Finin and A. Josh. Trust-Based Security in Pervasive Computing Environments. *IEEE Computer*, pages 154–157, December 2001. 1.2.1, 1.2.2, 2.3.1

[76] G. Karjoth, M. Schunter, and M. Waidner. Platform for Enterprise Privacy Practices: Privacy-enabled Management of Customer Data. In *Proceedings of 2nd Workshop on Privacy Enhancing Technologies (PET2002)*, pages 69–84, April 2002. 2.1, 2.2.2

[77] L. Kissner and D. Song. Privacy-Preserving Set Operations. In *Proceedings of CRYPTO 2005*, August 2005. 4.9

[78] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized Credential Translation: A Solution to Web Access Control. In *Proceedings of 10th Usenix Security Symposium*, August 2001. 4.2, 4.5.2, 4.9

[79] S. Lahlou and F. Jegou. European Disappearing Computer Privacy Design Guidelines V1.0. Ambient Agoras IST-DC report D15.4. Disappearing Computer Initiative, October 2003. 3.4

[80] S. Lahlou, M. Langheinrich, and C. Röcker. Privacy and Trust Issues with Invisible Computers. *Communications of the ACM*, 48(3):59–60, March 2005. 1, 2.1, 8.4

[81] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10(4): 263–310, November 1992. 2.3.1, 2.3.3, 2.5.2, 4.2, 4.4, 4.5.2, 4.9, 1

[82] M. Langheinrich. Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. In *Proceedings of Third International Conference on Ubiquitous Computing (Ubicomp 2001)*, pages 273–291, Oct 2001. 1, 2.1

[83] M. Langheinrich. A Privacy Awareness System for Ubiquitous Computing Environments. In *Proceedings of UbiComp 2002*, pages 237–245, September 2002. 2.1

[84] M. Langheinrich. The DC-Privacy Troubadour - Assessing Privacy Implications of DC-Projects. In *Proceedings of Designing for Privacy Workshop. DC Tales Conference*, June 2003. 8.4

[85] U. Leonhardt and J. Magee. Security Considerations for a Distributed Location Service. *Journal of Network and Systems Management*, 6(1):51–70, March 1998. 2.8.1

[86] N. Li, W. Du, and D. Boneh. Oblivious Signature-Based Envelope. In *Proceedings of 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, pages 182–189, July 2003. 6.6

[87] N. Li and J. C. Mitchell. RT: A Role-based Trust-managment Framework. In *Proceedings of The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212, April 2003. 2.5.1, 2.8.3

[88] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed Credential Chain Discovery in Trust Management. *Journal of Computer Security*, 11(1):35–86, February 2003. 2.8.4, 3.10, 8.3.1

[89] Shamus Software Ltd. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). `http://indigo.ie/~mscott/`. 6.5, 7.6, A.3, A.1

[90] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating Key Managment from File System Security. In *Proceedings of 17th ACM Symposium on Operating Systems Principles*, pages 124–139, December 1999. 1.1

[91] P. McDaniel. On Context in Authorization Policy. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 80–89, June 2003. 5.10

[92] Microsoft. Microsoft Windows 2000 Advanced Server. `http://www.microsoft.com/windows2000/en/advanced/help/`. 1.1

[93] K. Minami and D. Kotz. Secure Context-sensitive Authorization. In *Proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, pages 257–268, March 2005. 1.2.1, 1.2.3, 1.2.4, 2.8.2, 2.8.3, 5.10, 8.3.1

[94] K. Minami and D. Kotz. Secure Context-sensitive Authorization. *Journal of Pervasive and Mobile Computing (PMC)*, 1(1), March 2005. 5.10

[95] A. C. Myers and B. Liskov. Complete, Safe Information Flow with Decentralized Labels. In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, May 1998. 3.10

[96] G. Myles, A. Friday, and N. Davies. Preserving Privacy in Environments with Location-Based Applications. *Pervasive Computing*, 2(1):56–64, January-March 2003. 1.2.1, 2.3.1, 2.8.1, 2.8.3

[97] B.C. Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *Proceedings of International Conference on Distributed Computing Systems*, pages 283–291, May 1993. 4.2, 4.5.2, 4.9

[98] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 65–79, June 2003. 5.10

[99] Nextel. `http://www.nextel.com`. 1

[100] National Institute of Standards and Technology. Role-Based Access Control. American National Standard - ANSI INCITS 359-2004, February 2004. 1.1, 3, 3.2

[101] Oracle. Oracle Database 10g Security and Identity Managment. An Oracle White Paper, December 2003. 1.1, 1.1.2, 1.1.4

[102] J. Park and R. Sandhu. The UCON_{ABC} Usage Control Model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, February 2004. 5.10

[103] P. Persiano and I Visconti. User Privacy Issues Regarding Certificates and the TLS protocol. In *Proceedings of 7th ACM Conference on Computer and Communications Security (CCS 2000)*, pages 53–62, November 2000. 6.6

[104] F. Pfenning and C. Schürmann. System Description: Twelf - A Meta-Logical Framework for Deductive Systems. In *Proceedings of 16th International Conference on Automated Deduction (CADE-16-99)*, pages 202–206, July 1999. 3.6.3, 3.9.5

[105] OpenSSL Project. OpenSSL. `http://www.openssl.org`. 4.8.1

[106] F. Rabitti, E. Bertino, W. Kim, and Darrell Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Transactions on Database Systems*, 16 (1):88–131, March 1991. 3.10

[107] I. Ray, I. Ray, and N. Narasimhamurthi. A Cryptographic Solution to Implement Access Control in a Hierarchy and More. In *Proceedings of 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pages 65–73, June 2002. 6.4.1, 7.7

[108] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975. 2.4.3, 3.5.2

[109] R. S. Sandhu. Cryptographic Implementation of a Tree Hierarchy for Access Control. *Information Processing Letters*, 27(2):95–98, 1988. 7.7

[110] K. E. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies During Automated Trust Negotation. In *Proceedings of Network and Distributed System Security Symposium (NDSS'01)*, February 2001. 6.6

[111] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Proceedings of Crypto '84*, pages 47–53, 1984. 6.3.1

[112] A. T. Sherman and D. A. McGrew. Key Establishment in Large Dynamic Groups. *IEEE Transactions on Software Engineering*, 29(5):444–458, May 2003. 7.7

[113] N. P. Smart. Access Control Using Paring Based Cryptography. In *Proceedings of The Cryptographer's Track at RSA Conference (CT-RSA 2003)*, pages 111–121, April 2003. 7.7

[114] D. K. Smetters and G. Durfee. Domain-Based Administration of Identity-Based Cryptosystems for Secure Email and IPSEC. In *Proceedings of 12th Usenix Security Symposium*, August 2003. 6.4.1, 6.5

[115] K. R. Sollins. Cascaded Authentication. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 156–163, May 1988. 4.2, 4.5.2, 4.9

[116] D. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, May 2000. 4.9

[117] M. Spreitzer and M. Theimer. Providing Location Information in a Ubiquitous Computing Environment. In *Proceedings of SIGOPS '93*, pages 270–283, Dec 1993. 2.8.1

[118] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the Winter 1988 USENIX Conference*, pages 191–202, February 1998. 1.1

[119] Claymore Systems. PureTLS. `http://www.rtfm.com/puretls/`. 2.6.4

[120] R. Tamassia, D. Yao, and W. H. Winsborough. Role-Based Cascaded Delegation. In *Proceedings of 9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 146–155, June 2004. 8.3.1

[121] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka. Context-Based Secure Resource Access in Pervasive Computing Environments. In *Proceedings of First IEEE International Workshop on Pervasive Computing and Communications Security(PerSec'04)*, pages 159–163, March 2004. 1.2.1, 2.3.1

[122] W.-G. Tzeng. A Time-Bound Cryptographic Key Assignment Scheme for Access Control in a Hierarchy. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):182–188, 2002. 7.7

[123] VERSUSTECH. `http://www.versustech.com`. 4.8.1

[124] D. M. Wallner, E. J. Harder, and R. C. Agee. Key Management for Multicast: Issues and Architectures. RFC 2627, June 1999. 7.7

[125] B. R. Waters. Efficient Identity-Based Encryption Without Random Oracles. In *Eurocrypt 2005*, May 2005. 6.5

[126] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an Encrypted and Searchable Audit Log. In *Proceedings of 11th Annual Network and Distributed System Security Symposium (NDSS 2004)*, February 2004. 6.5

[127] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991. 1

[128] V. Welch, I. Foster, C. Kesselman, O. Mulmu, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *Proceedings of 3rd Annual PKI R&D Workshop*, April 2004. 2.8.4

[129] W. H. Winsborough and N. Li. Towards Practical Automated Trust Negotiation. In *Proceedings of 3rd International Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, pages 92–103, June 2002. 6.6

[130] W. H. Winsborough and N. Li. Safety in Automated Trust Negotiation. In *Proceedings of 2004 IEEE Symposium on Security and Privacy*, pages 147–160, May 2004. 5.10, 6.6

[131] C. K. Wong, M. Gouda, and S. Lam. Secure Group Communications using Key Graphs. In *Proceedings of ACM SIGCOMM '98*, pages 68–79, September 1998. 7.7

[132] C. Wullems, M. Looi, and A. Clark. Towards Context-aware Security: An Authorization Architecture for Intranet Environments. In *Proceedings of First IEEE International Workshop on Pervasive Computing and Communications Security(PerSec'04)*, pages 132–137, March 2004. 1.2.1, 2.3.1

[133] D. Yao, Y. Dodis, N. Fazio, and A. Lysyanskaya. ID-Based Encryption for Complex Hierarchies with Applications to Forward Security and Broadcast Encryption. In *Proceedings of 11th ACM Conference on Computer and Communications Security (CCS 2004)*, pages 354–363, October 2004. 6.4.3, 6.5, 7.3

[134] T. Yu and M. Winslett. A Unified Scheme for Resource Protection in Automated Trust Negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 110–122, May 2003. 6.6

[135] Y. Zheng, T. Hardjono, and J. Seberry. New Solutions to the Problem of Access Control in a Hierarchy. Technical Report Preprint No.93-2, Department of Computer Science, University of Wollongong, Australia, 1993. 7.7